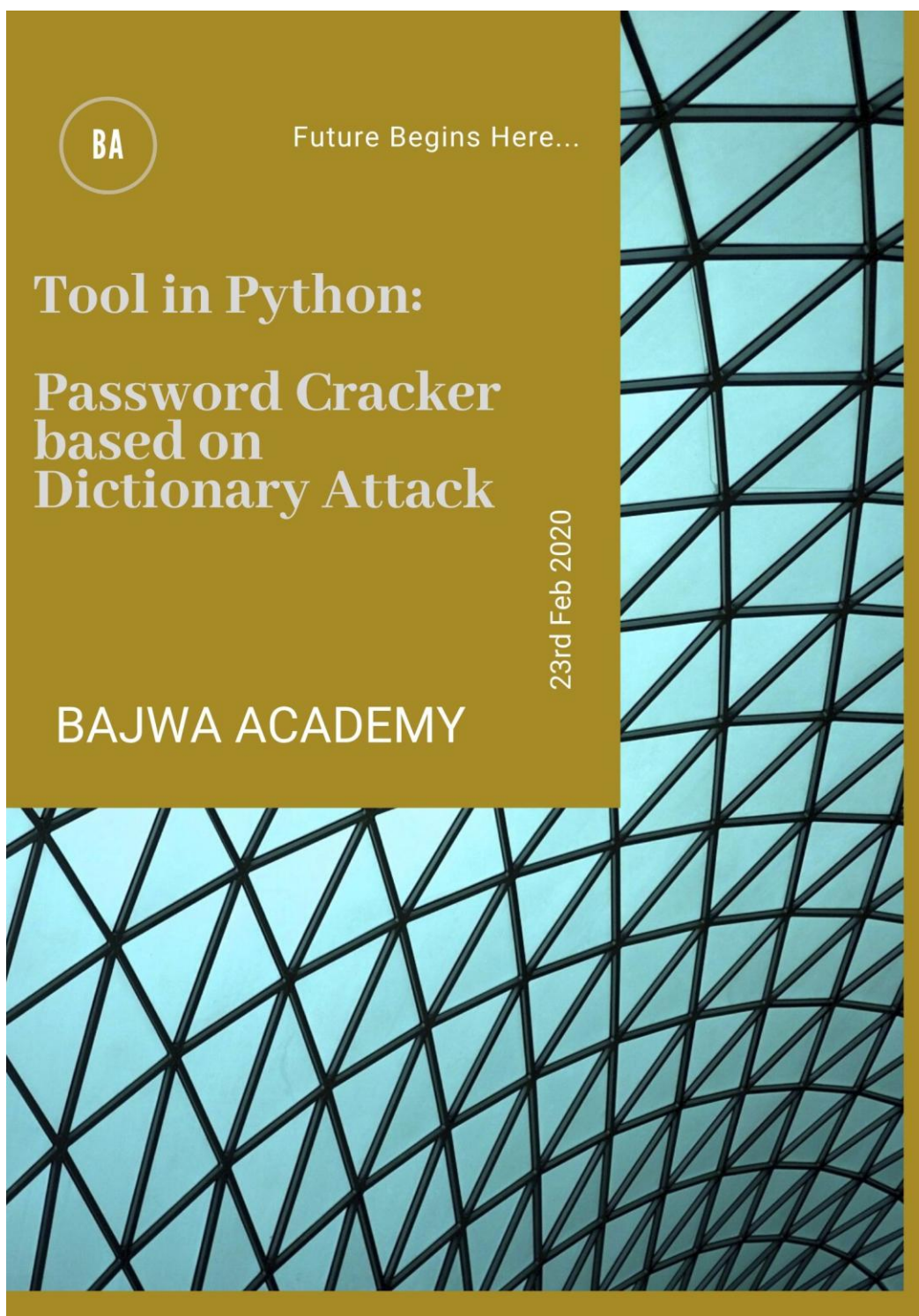


## Tool in Python: Password Cracker Based on Dictionary Attack

Dilpreet Singh Bajwa



## **Tool in Python: Password Cracker based on Dictionary Attack**

**Software Requirement:** Windows Machine, Pycharm or any other IDE.

**Hardware Requirement:** Laptop or PC (i3 or i5, 2 GB RAM).

**Prerequisite Knowledge:** Basic Programming logics, OOPs concept and Python Basic Level.

### **Steps for Creation of Tool in Python:**

**Step1:** Install “python” and “pycharm” on your windows machine. You may also use virtual windows machine. You may download Python from <https://www.python.org/downloads/> and pycharm from <https://www.jetbrains.com/pycharm/>.

**Step2:** Use pycharm to write, edit and execute your tool.

**Here first we provide background information about basic password cracking techniques and afterward focus on only on step-2 i.e. we code and execute the tool and share the code and its explanation with you.**

**Introduction:** Password usage is part of our daily walk of life; we are using passwords daily to unlock our computer system or mobile phone. We use passwords for our email accounts, banking accounts, social networking sites accounts (like Facebook), professional networking sites accounts (like LinkedIn) and other different logins on different sites for various purposes.

But you must be aware of the fact that passwords are the most vulnerable part of any security mechanism, if not properly chosen, created or dealt with. It can be guessed or cracked through various available techniques/tools and provides complete unauthorized access to attackers or hackers of all your confidential information.

In the Linux operating system, passwords are stored in the /etc/shadow file, while in Windows; the passwords are stored in the SAM file. Hacker may compromise these machines through various attacks or through malware and able to access these files but the password stored in these files are in hash form. In normal scenario, these files can be accessed only by the user having root privileges.

Generally, passwords are not saved in computer systems or servers as plain-text. The passwords are saved as a digest or as a hash value after applying one of the various hash algorithms available like md5, sha-1, sha-256, or sha-512. Hashing converts password into an encoded form called the hash password by applying a one-way hash function. “One-way” means that it is practically impossible to convert the hash password back to the original password in plain text. Hashes are always unique corresponding to particular data. Change in even a single bit of data produces a different hash.

**For example:**

Plain Text Password: “**Administrator@123**” and its corresponding hash values are given below by applying different hash algorithms:

**MD5 Hash Value:**

6ba2f6e7cc59e4538b63048c3a28cbef

**SHA1 Hash Value:**

d978e5b012e8967a9f5b86251e54ae77b5e924c3

**SHA256 Hash Value:**

8a768f4ba64d9492a67828c66cee26fec0129c07e599ee9ecb1f57520ba6797a

**SHA512 Hash Value:**

6cd2852eebaf9cf19f7331656700a540e469d112fa8142402086c34691e84b943a6d760e81f36fdc31ef8bb710be4f7378e70a393a2faa0732c05fbc9cec91ab

There are various techniques and tools available to crack the passwords. There are various attacks also through which you may extract the password of the victim. We may not go in that

detail. Here we are focusing on basic password cracking techniques mentioned below and after that, we write code in python for Password Cracker based on Dictionary attack:

- a) **Brute-Force Attack**
- b) **Dictionary Attack**
- c) **Rainbow Table Attack**

- a) **Brute-Force Attack:** Brute force is your last resort for password cracking as it requires more effort in terms of processing power and time while cracking the password. The time and processing power it takes depends upon the password strength. The more computing power you have, there are more chances that you may be more successful with this technique and it always works if you have sufficient resources like one or more computers or a supercomputer. In this technique, we try all possibilities of all alphabetic characters-lower or upper case both, numbers, special characters that might be used for the password.
- b) **Dictionary Attack:** Dictionary attack is simple, fastest and is one of the first approaches you may try to crack any password. Here we have a predefined list of words or possible passwords called as dictionary and we may try each one by one until a password is matched or list exhaust. The list may contain millions of passwords/words and manually it is really a daunting task to try such a huge list of passwords but computers manage the work easily and compare millions of words in hours. Your success depends upon the cautious collection of words in the list and if your password is short or weak then this attack may crack it in a few seconds.
- c) **Rainbow Table Attack:** As we told you earlier that now a day due to security concerns passwords cannot store as plain text but rather as hashes. One technique is based on the dictionary attack mentioned above (for which we also written code in this article), it takes a predefined list of passwords/words, calculate hash of each word compare it to the victim's hashed password until either hash matched or list exhausts. Another technique is to create a table having all the words of the dictionary already hashed and compare your password hash with these hashes in the table until you know the password or list ends. It is a faster approach compared to a dictionary attack.

**As you understand about basic password cracking techniques, now we will focus on our main topic that is to create a password cracking tool based on Dictionary Attack in python and explain its code.**

Our program works like this, it provides us menu to choose from various options that which type of hash password you want to crack like md5, sha1, sha256, etc. and user choose the option accordingly and provide the hashed password (which is not understandable by user because it is in hash form of plain text password) to the program. After that, the tool asks the user to provide the dictionary file which is actually the list of probable passwords in plain text. The list may contain only a hundred words or may contain millions. The more the list is exhaustive, there are more chances that you may crack the password. After that our program picks each plain text password/word from this dictionary file, calculates its hash, compares it with our hashed password provided by the user. This whole process of picking the word, calculating the hash, and comparing it, is iterative in nature and repeats until either calculated hash matched with hashed password or our dictionary list ends. If we found the match, we will display that “Password Found” and corresponding plain-text password on the screen otherwise we display that “Sorry password Not Found”.



The code is given below:

\*\*\*\*\*CODE\*\*\*\*\*

```

3     import hashlib
4
5
6     def input_hash():
7         while True:
8             print("This is Password Cracker Tool based on dictionary attack")
9             print('Press "1" to crack "md5" hash password-----:')
10            print('Press "2" to crack "sha1" hash password-----:')
11            print('Press "3" to crack "sha224" hash password-----:')
12            print('Press "4" to crack "sha256" hash password-----:')
13            print('Press "6" to crack "sha512" hash password-----:')
14            print('Press "7" to "Quit"-----:')
15            choice = input("Enter Choice: 1 or 2 or 3 or 4 or 5 or 6 or 7: ")
16            print(choice)
17            if choice == "1":
18                hash_password = input("Enter md5 hash: ")
19                break
20            elif choice == "2":
21                hash_password = input("Enter sha1 hash: ")
22                break
23            elif choice == "3":
24                hash_password = input("Enter sha224 hash: ")
25                break
26            elif choice == "4":
27                hash_password = input("Enter sha256 hash: ")
28                break
29            elif choice == "5":
30                hash_password = input("Enter sha384 hash: ")
31                break
32            elif choice == "6":
33                hash_password = input("Enter sha512 hash: ")
34                break
35            elif choice == "7":
36                quit()
37
38        return hash_password, choice

```

```

39
40
41 def password_cracker():
42     flag = 0
43     pass_hash, algorithm = input_hash()
44     word_file = input("Enter File name for Word List: ")
45     try:
46         word_file_open = open(word_file, "r")
47     except:
48         print("File not Found")
49         quit()
50
51     for each_word in word_file_open:
52         enc_word = each_word.encode('utf-8')
53         if algorithm == "1":
54             digest = hashlib.md5(enc_word.strip()).hexdigest()
55         elif algorithm == "2":
56             digest = hashlib.sha1(enc_word.strip()).hexdigest()
57         elif algorithm == "3":
58             digest = hashlib.sha224(enc_word.strip()).hexdigest()
59         elif algorithm == "4":
60             digest = hashlib.sha256(enc_word.strip()).hexdigest()
61         elif algorithm == "5":
62             digest = hashlib.sha384(enc_word.strip()).hexdigest()
63         elif algorithm == "6":
64             digest = hashlib.sha512(enc_word.strip()).hexdigest()
65
66         if digest == pass_hash:
67             print(f"Given password hash = {pass_hash}")
68             print("Digest Calculated = " + digest)
69             print("Congratulations Hash Matched----- PASSWORD FOUND")
70             print(f"Password is: {each_word} ")
71             flag = 1
72             break
73     if flag == 0:
74         print("Sorry, Password Not Found")
75
76
77 password_cracker()

```

## Explanation:

### Line3:

```
3 import hashlib
```

Firstly, we import the most important module of our program “hashlib” as specified above. “hashlib” is an inbuilt module and provide us many secure hash or message digest functions for calculating hash such as SHA1, SHA224, SHA256, SHA512, and MD5. All functions return a hash object or simply hash corresponding to the parameter we pass to the function.

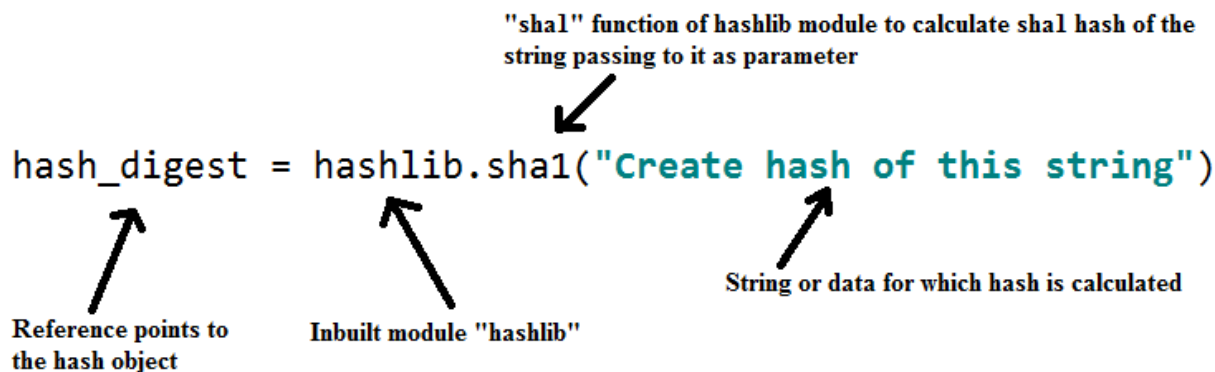
For example, we may use sha1() to create a SHA1 hash object as shown below:

"sha1" function of hashlib module to calculate sha1 hash of the  
string passing to it as parameter

↓

```
hash_digest = hashlib.sha1("Create hash of this string")
```

Reference points to the hash object      Inbuilt module "hashlib"      String or data for which hash is calculated



### Line 9-41: see snapshot given below:

We define any function in python by using the following syntax:

```
def function_name():
```

```
    statement-1
```

```
    statement-2
```

```
    .
```

```
    .
```

```
    statement-n
```



Where def is the keyword to define a function, after that we use function name along with parenthesis (in our case its “input\_hash()”), inside the function we may write one or more statements which may be executed when we call the function called as a function body.

```

9  def input_hash():
10     while True:
11         print("This is Password Cracker Tool based on dictionary attack")
12         print('Press "1" to crack "md5" hash password-----:')
13         print('Press "2" to crack "sha1" hash password-----:')
14         print('Press "3" to crack "sha224" hash password-----:')
15         print('Press "4" to crack "sha256" hash password-----:')
16         print('Press "6" to crack "sha512" hash password-----:')
17         print('Press "7" to "Quit"-----:')
18         choice = input("Enter Choice: 1 or 2 or 3 or 4 or 5 or 6 or 7: ")
19         print(choice)
20         if choice == "1":
21             hash_password = input("Enter md5 hash: ")
22             break
23         elif choice == "2":
24             hash_password = input("Enter sha1 hash: ")
25             break
26         elif choice == "3":
27             hash_password = input("Enter sha224 hash: ")
28             break
29         elif choice == "4":
30             hash_password = input("Enter sha256 hash: ")
31             break
32         elif choice == "5":
33             hash_password = input("Enter sha384 hash: ")
34             break
35         elif choice == "6":
36             hash_password = input("Enter sha512 hash: ")
37             break
38         elif choice == "7":
39             quit()
40
41     return hash_password, choice

```

while loop is executing always and displays user with no. of choices to choose any one option 1-7. It asks user to tell which type of hash password you want to crack md5, sha1, sha512 etc. and last option 7 is to quit from program.

Here, we are taking input from user specifying which option user selected from above choice i.e 1-7. According to that we proceed or call appropriate hash function.

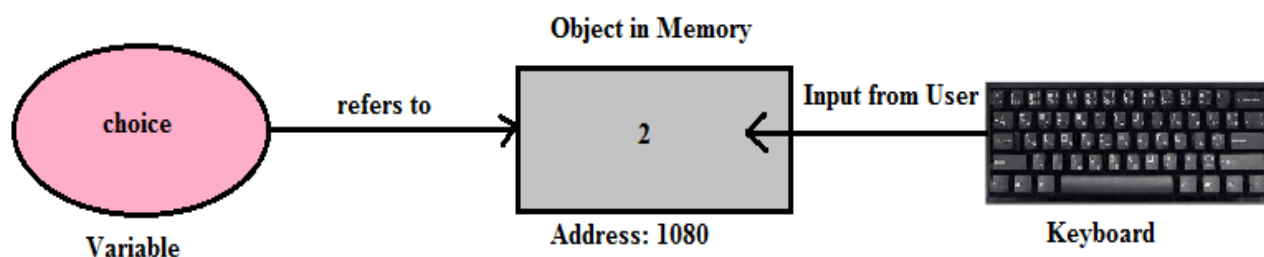
Line 20-37, Options 1-6,  
Here we take input from user that is the hashed password corresponding to which we have to find the plain-text password. In line 18, user provides the choice that is which hashed password(md5, sha1 etc.) he/she is going to provide and now as per his/her choice corresponding "if" block statement executes for example user choose the option 2 i.e the sha1 hashed password, so line 23 "if" statement satisfies the condition and the statement "hash\_password = input("Enter sha1 hash: ") "executes which takes input from the user which is a sha1 hashed password, stores it in the memory and the reference or variable "hash\_password" points to it or simply we say that whenever we use "hash\_password" variable in program that means we are talking about the hashed password provided by the user for cracking.

If the user choose the option 7 at line 18 then this "if" statement at line 38 executes and the we exit from the program

Here at Line 41, the function "input\_hash" returns two values: hash\_password (that refers to hashed password value provided by user) and choice (that refers to the choice chosen by user i.e the type of hashed password (sha1, md5, sha512 etc.)

Here we have defined a function named “input\_hash” that provides options(1-6) to the user and asks to choose what type of hash password user wants to crack. For example, if the user chooses the option ” 2” means the user wants to crack a password hashed with sha1 algorithm. In line 18, we are using a variable “choice” (you may give any name as per rules for python language naming convention of identifiers but the name should be meaningful).

“choice” is actually a reference and refers to a String object that points to option “2” taken from the user through the input() function. Here input("Enter choice 1, 2, 3, 4, 5, 6 or 7: ") function prints the message "Enter choice 1, 2, 3, 4, 5, 6 or 7: " on screen and waits for the user to provide the choice which stores in the memory as String object and the variable "choice" refer to it. So in line 18 user provides the input through input() function and the “choice” variable refers to the option provided by the user (in our case it's 2) as depicted in the figure given below:



From line 20 to 38 series of “if-elif” blocks defined and executes based on the value refers by choice variable. As we know that the user provides the value “2” in our case so the corresponding “elif” block at line 23 satisfies the condition and executes.

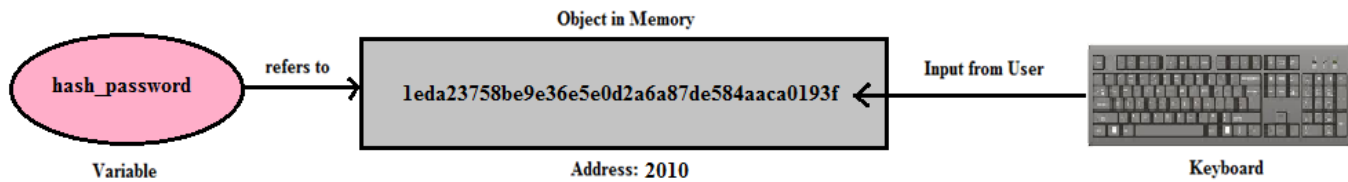
```

23 elif choice == "2":
24     hash_password = input("Enter sha1 hash: ")
25     break

```

Similarly, if the user chooses some another option for example 1, 3, 4, 5, 6, or 7 then corresponding “if” or “elif” block executes.

When “choice == 2” condition satisfies at line 23 then all statements of this “elif” block executes. It has only two statements in line 24 and 25. The statement at line 24 is again a user input statement where the program asks for hash password user wants to crack. The user provides the hashed password through the “input()” function and the variable “hash\_password” refers to it.



In the next statement “break” will execute and take control out of “while” loop to the line 41 that is the last statement of the “input\_hash” function where we are returning two values through return statement. Here we are returning the “choice” and “hash\_password” variables that contain the information about which type of hashed password (sha1 in our case) we provide and the corresponding hashed password (1eda23758be9e36e5e0d2a6a87de584aaca0193f) that we are going to crack.

**Line 44-77: see snapshot given below:**

Now in line 44 to 77, we have defined our main function “password cracker” that contains the main logic of the program. Here we try to crack the hashed password provided by the user by using “Dictionary Attack”.

In this function, we take “dictionary file” that contains possible list of passwords in plain-text. We take each password from this file one by one, calculate its hash, and compare this calculated hash with our hashed password (1eda23758be9e36e5e0d2a6a87de584aaca0193f) provided by the user until the match found or the dictionary file contains no more plain-text passwords. This whole process is iterative in nature, implemented by using a “for” loop. Now let us elaborate the whole function line by line.

**At line 45,** we are using a variable named “flag” and assigned it a value 0. We use it as a flag to check whether we are successful in cracking the password or not. Initially its value is zero and when we crack the password we assign it a new non-zero value say 1 and check it inside our program (see line 76) whether its value is zero or not, zero means password not cracked or found so that we can display an appropriate message for the user.

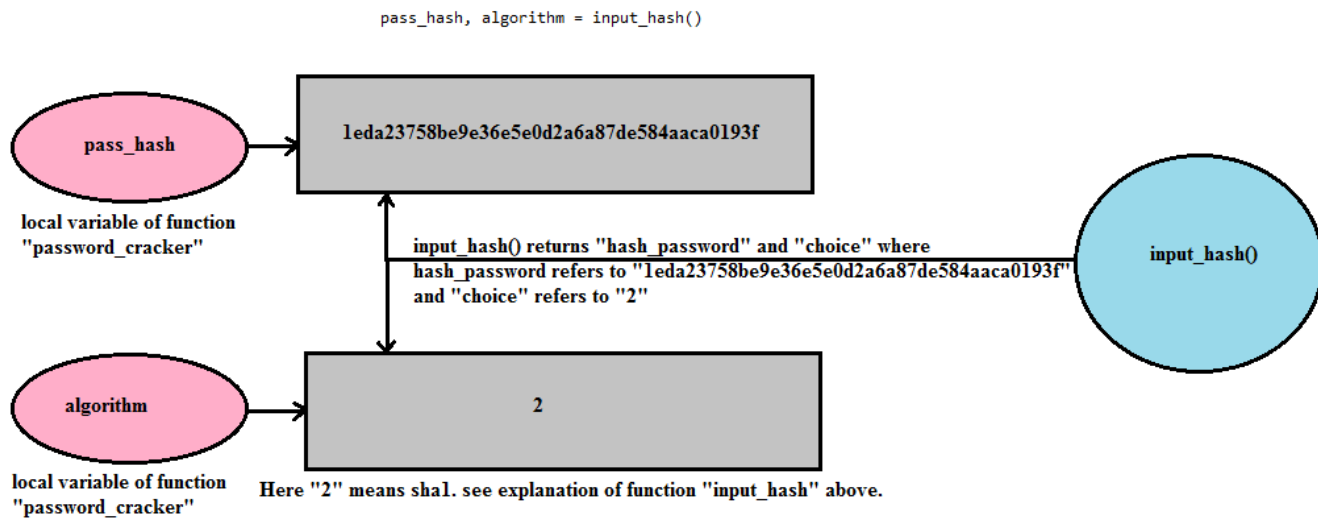
```

44 def password_cracker():
45     flag = 0
46     pass_hash, algorithm = input_hash()
47     word_file = input("Enter File name for Word List: ")
48     try:
49         word_file_open = open(word_file, "r")
50     except:
51         print("File not Found")
52         quit()
53
54     for each_word in word_file_open:
55         enc_word = each_word.encode('utf-8')
56         if algorithm == "1":
57             digest=hashlib.md5(enc_word.strip()).hexdigest()
58         elif algorithm == "2":
59             digest = hashlib.sha1(enc_word.strip()).hexdigest()
60         elif algorithm == "3":
61             digest = hashlib.sha224(enc_word.strip()).hexdigest()
62         elif algorithm == "4":
63             digest=hashlib.sha256(enc_word.strip()).hexdigest()
64         elif algorithm == "5":
65             digest=hashlib.sha384(enc_word.strip()).hexdigest()
66         elif algorithm == "6":
67             digest=hashlib.sha512(enc_word.strip()).hexdigest()
68
69         if digest == pass_hash:
70             print(f"Given password hash = {pass_hash}")
71             print("Digest Calculated = " + digest)
72             print("Congratulations Hash Matched----- PASSWORD FOUND")
73             print(f"Password is: {each_word} ")
74             flag = 1
75             break
76     if flag == 0:
77         print("Sorry, Password Not Found")
78

```

At line: 46: 46 pass\_hash, algorithm = input\_hash()

Here we are calling the “input\_hash” function discussed above which is returning two values, one is “hashed password” we have to crack and the “choice” given by the user specifying which algorithm used to hash it. Both these values return by “input\_hash” function are now referred by two new local variables “pass\_hash” and “algorithm” of “password\_cracker” function as shown below:



**Lines 47-52:**

```

47 word_file = input("Enter File name for Word List: ")
48 try:
49     word_file_open = open(word_file,"r")
50 except FileNotFoundError:
51     print("File not Found")
52     quit()

```

In the above lines, the program asks the user about the name of the dictionary file which he/she wants to use for password cracking. You may either specify only file name if file exists in the current directory or you have to specify the full path to the file. After that, we open the file (line 49) and if file not found then the system generate “FileNotFoundError” exception which we handle through exception handling mechanism “try-except” and when error occurs inside “try” block means when file not found then “except” block executes and displays “File not Found” message and quit from the program by calling “quit()” function.

To open the file, we are using an “open()” inbuilt function (at line 49) and passing two parameters to it. The first parameter can be the name of the file or any variable refers to the file name. Here, in our case, we pass the variable “word\_file” which is a variable and refers to the

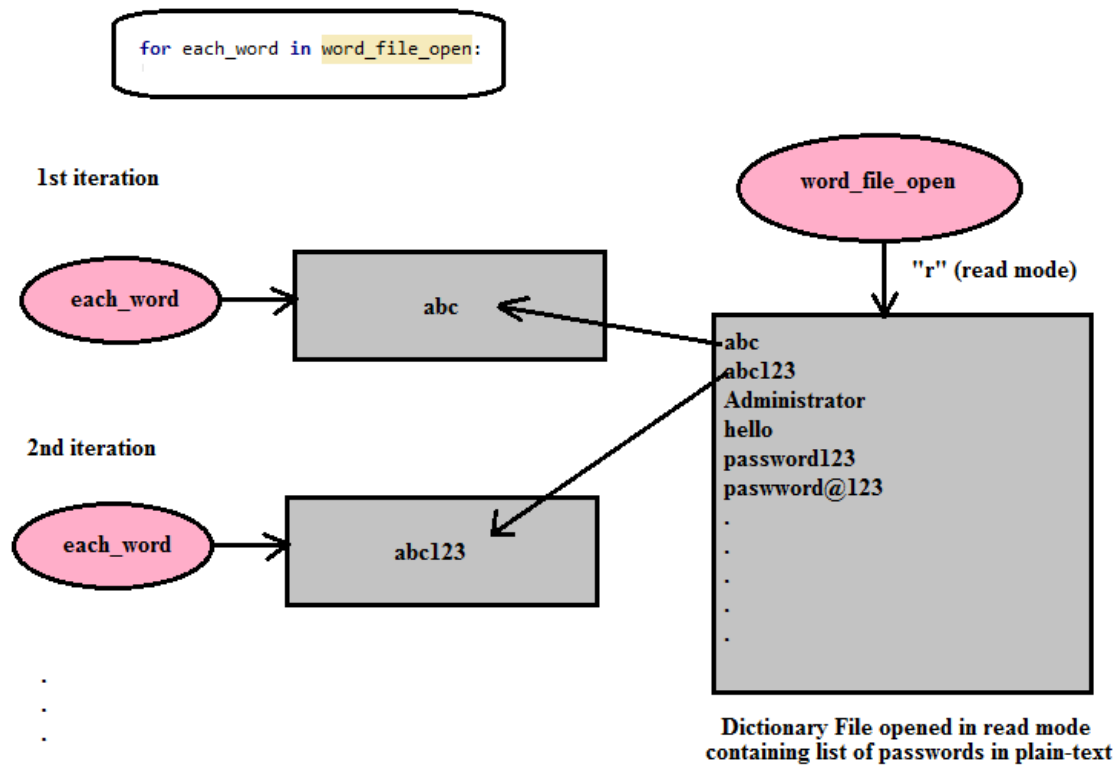
filename (dictionary file) provided by the user (at line 47). The second argument tells the open() function, in which mode file can be open. There are various modes like “r” for only reading, “w” for writing, “r+” for reading and writing and so on. We have opened the file in read mode.

**Line 54-67:** In these lines of code we implement “for” loop.

```
54 for each_word in word_file_open:
55     enc_word = each_word.encode('utf-8')
56     if algorithm == "1":
57         digest=hashlib.md5(enc_word.strip()).hexdigest()
58     elif algorithm == "2":
59         digest = hashlib.sha1(enc_word.strip()).hexdigest()
60     elif algorithm == "3":
61         digest = hashlib.sha224(enc_word.strip()).hexdigest()
62     elif algorithm == "4":
63         digest=hashlib.sha256(enc_word.strip()).hexdigest()
64     elif algorithm == "5":
65         digest=hashlib.sha384(enc_word.strip()).hexdigest()
66     elif algorithm == "6":
67         digest=hashlib.sha512(enc_word.strip()).hexdigest()
```

In “for” loop “each\_word” variable refers to each element or word from dictionary file (referred by “word\_file\_open”) one by one with each iteration of loop as depicted in the figure given below:





After that in **line 55**, the word or plain-text password we pick from the dictionary file which is now referred by “each\_word” variable, we encode it through the “encode()” function because it converts the string to bytes acceptable by hash function. The hash function accepts byte form data. This byte form data is now referred by new local variable “enc\_word”. The complete statement at line 55 is shown below:

```

55 enc_word = each_word.encode('utf-8')

```

In **Line 56-67** we specify a series of “if” and “elif” blocks that can be executed as per the condition. As you know we choose option “2” for hash password means the password we want to crack is encoded with sha1 algorithm. The value “2” now currently referred by the local variable “algorithm” in the function “password\_cracker” So to compare each plain-text password from dictionary file, we also have to convert it into sha1 hash. So as per option chosen by user corresponding “if” or “elif” block gets executed.

```

56 | if algorithm == "1":
57 |     digest=hashlib.md5(enc_word.strip()).hexdigest()
58 | elif algorithm == "2":
59 |     digest = hashlib.sha1(enc_word.strip()).hexdigest()
60 | elif algorithm == "3":
61 |     digest = hashlib.sha224(enc_word.strip()).hexdigest()
62 | elif algorithm == "4":
63 |     digest=hashlib.sha256(enc_word.strip()).hexdigest()
64 | elif algorithm == "5":
65 |     digest=hashlib.sha384(enc_word.strip()).hexdigest()
66 | elif algorithm == "6":
67 |     digest=hashlib.sha512(enc_word.strip()).hexdigest()

```

In our case following block of code executed as per the user choice “2”:

```

58 | elif algorithm == "2":
59 |     digest = hashlib.sha1(enc_word.strip()).hexdigest()

```

Now let’s understand the above line (59) of code to know how hash can be calculated. Here “hashlib” is inbuilt module, “sha1()” which is called with “.” operator is the function of this “hashlib” module to calculate sha1 hash of byte form data which is currently referred by enc\_word and to strip any unwanted leading or trailing spaces, we used “strip()” function along with it. Now the complete statement “**hashlib.sha1(enc\_word.strip())**” calculate and return sha1 hash of byte form data referred by “enc\_word” and at last to convert this result into hexadecimal format, we used and call “hexdigest()” function over it. This calculated hash now referred by the “digest” named variable.

**Next lines 66-72** checks by using “if” condition that the calculated hash which is now referred by the “digest” variable is equal to the hash value provided by the user at the start of the program which is currently referred by “pass\_hash” variable in this function. If both matches that mean we found the password, all statements of “if” block get executes. So we display password hash value provided by the user, calculated hash value and corresponding plain-text password by using print() function, change the value of flag variable to non-zero value “1” which signifies that password is found and at last by using “break” to come out of “for” loop because there is no need to check other plain-text passwords from dictionary file.

```
66 | if digest == pass_hash:
67 |     print(f"Given password hash = {pass_hash}")
68 |     print("Digest Calculated = " + digest)
69 |     print("Congratulations Hash Matched----- PASSWORD FOUND")
70 |     print(f"Password is: {each_word} ")
71 |     flag = 1
72 |     break
```

Now last two statements of function “password\_cracker” are given below:

```
73 | if flag == 0:
74 |     print("Sorry, Password Not Found")
```

Here we check whether the “flag” variable value is zero or non-zero after coming out of the “for” loop. If value is non-zero then corresponding “if” block statement can’t execute because password already found and we assign a non-zero value to “flag” (line 71 above) but if flag value is still zero that means after checking all passwords from dictionary file through “for” loop, we can’t be able to find or match the password so we at last display through this “if” block that “Sorry, Password Not Found”.

**Line 77:** 77 password\_cracker()

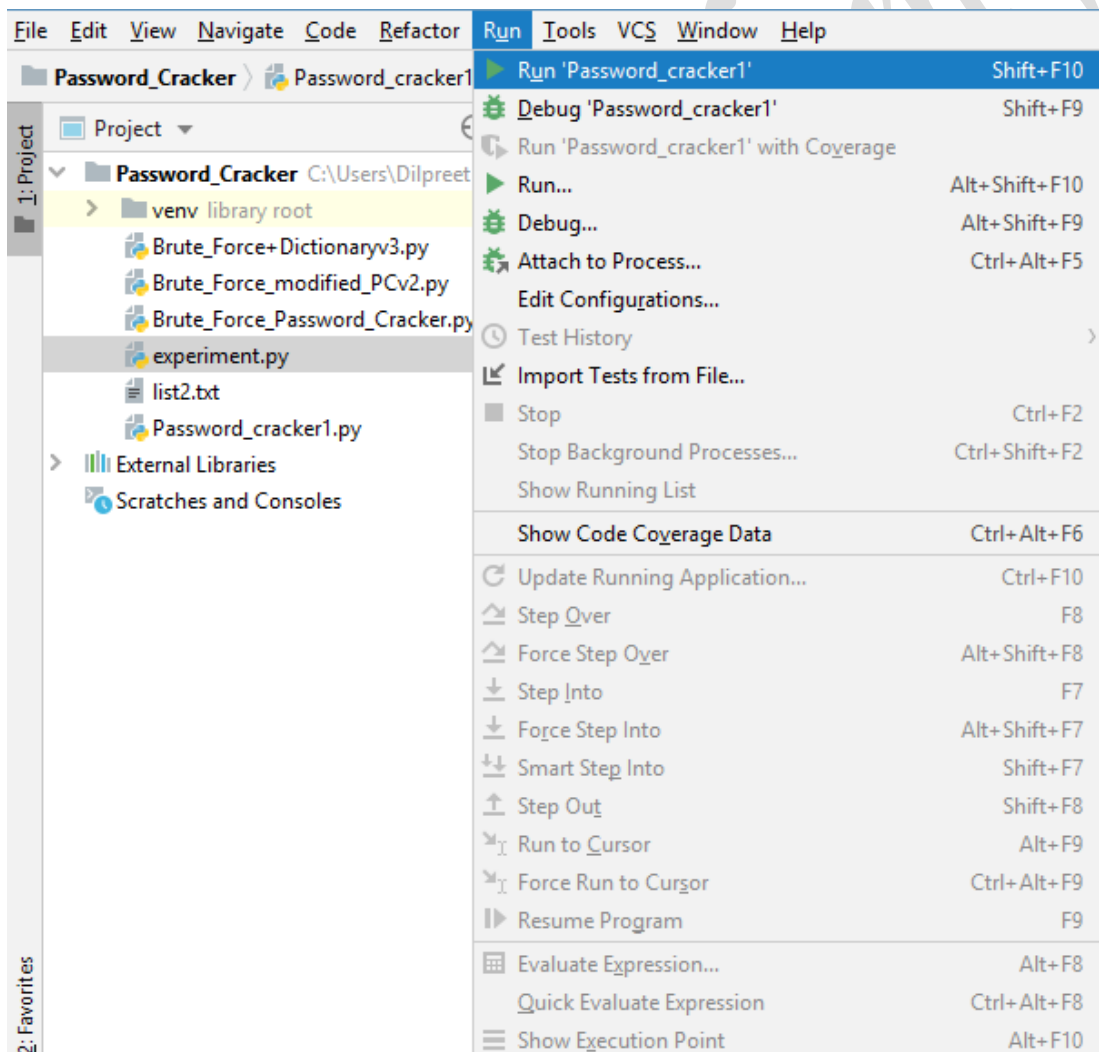
Above we import “hashlib” module, define the “input\_hash” function and then define the “password\_cracker” function. Just defining any function is not confirms the execution of its code. To execute the function, we have to call it. In line 77, we are calling the function “password\_cracker()” which in turn takes control to the definition of function and executes all statements of the function as per logic defined there.

## Execution of Program:

Above we have discussed and defined all code corresponding to our “password cracker” tool based on dictionary attack. All code is written in pycharm and the file is saved as “password\_cracker1.py” python file. Now we will see how to execute the code using pycharm and what is its output.

To execute follow the steps given below:

Go to: Pycharm>Menu>Run>Run.. and execute the program:



## Output:

```
This is Password Cracker Tool based on dictionary attack
Press "1" to crack "md5" hash password-----:
Press "2" to crack "sha1" hash password-----:
Press "3" to crack "sha224" hash password-----:
Press "4" to crack "sha256" hash password-----:
Press "6" to crack "sha512" hash password-----:
Press "7" to "Quit"-----:
Enter Choice: 1 or 2 or 3 or 4 or 5 or 6 or 7:  2
2
Enter sha1 hash: 1eda23758be9e36e5e0d2a6a87de584aaca0193f
Enter File name for Word List: list2.txt
Given password hash = 1eda23758be9e36e5e0d2a6a87de584aaca0193f
Digest Calculated = 1eda23758be9e36e5e0d2a6a87de584aaca0193f
Congratulations Hash Matched----- PASSWORD FOUND
Password is: Administrator
```

```
Process finished with exit code 0
```