**ALGORITHM:**

1. Get a set of random words W. Each word is a string with no whitespaces.
2. Add k numbers and/or symbols to random positions of each word. Add each iteration of this process to the set W. This generates k\*|W| new strings
3. Multiply the set. This essentially transforms to performing a limited version the regular operation star on the set. This generates |W|\*m new strings by appending m random strings from W to each string. Let the set of these newly generated strings be M.

For each word w in W,

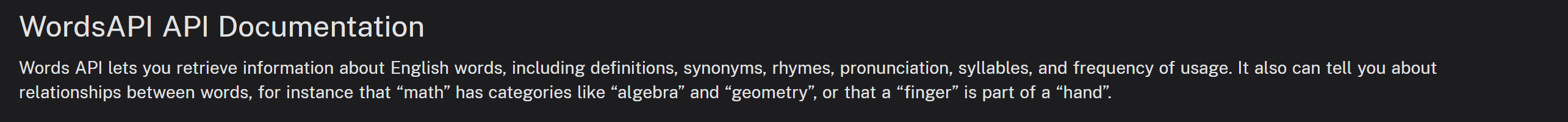
1. Let x = w
2. For i in range(m) where m is the multiplier
3. choose a random word ri and append it to x.
4. add the new string to the set M.
5. x = new\_string
6. Now write M U W (union) to a file.

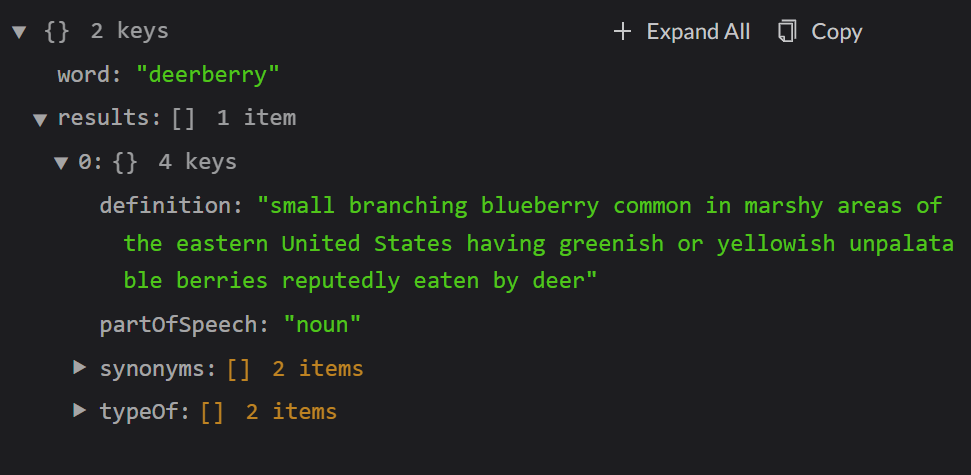
**IMPLEMENTATION**

Technologies used:

* Python v3.11
* Libraries = requests, dotenv, os.
* API = WordsAPI (<https://rapidapi.com/dpventures/api/wordsapi/>)

**Step 1: Fetching the words**

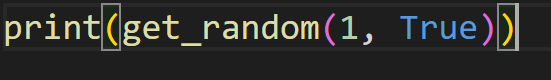


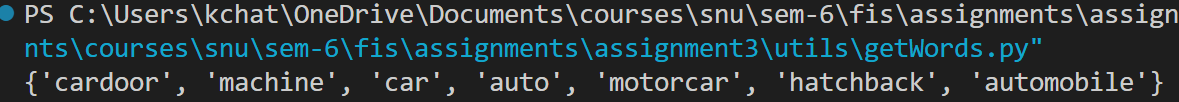
This API has a route called words/random which returns the request with a random word and the various categories that word falls in.

For example, this is a sample response:

I wrote a python module called getWords.py which contains a function get\_random(n, flag). This function accepts an integer n and returns a set of n random words and their categories. It fetches n random words from the API. The inclusivity of the categories is dependent on the flag. All whitespaces are removed from the strings. No words are repeated.

For example:





**Step 2: Randomizing the words with numbers or/and symbols**

I wrote a python module called passwordGenerator.py that contains the function generate\_passwords and helper functions randomize\_string and generate\_string.

* generate\_string: (string: str, position: int, character: str) -> str

This function receives a string, a position, and a character and inserts the character at that position in the string. It returns the new string.

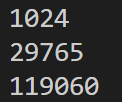
* randomize\_string: (string: str, type: str, count: int, count\_alt: int = 0) -> Array(str)

This function randomizes the string by adding the specified type of characters randomly to the string at random positions.

* generate\_passwords: (word\_set: set, multiply: bool, multiplier: int, numbers: bool, symbols: bool, max\_chars\_added: int) -> None

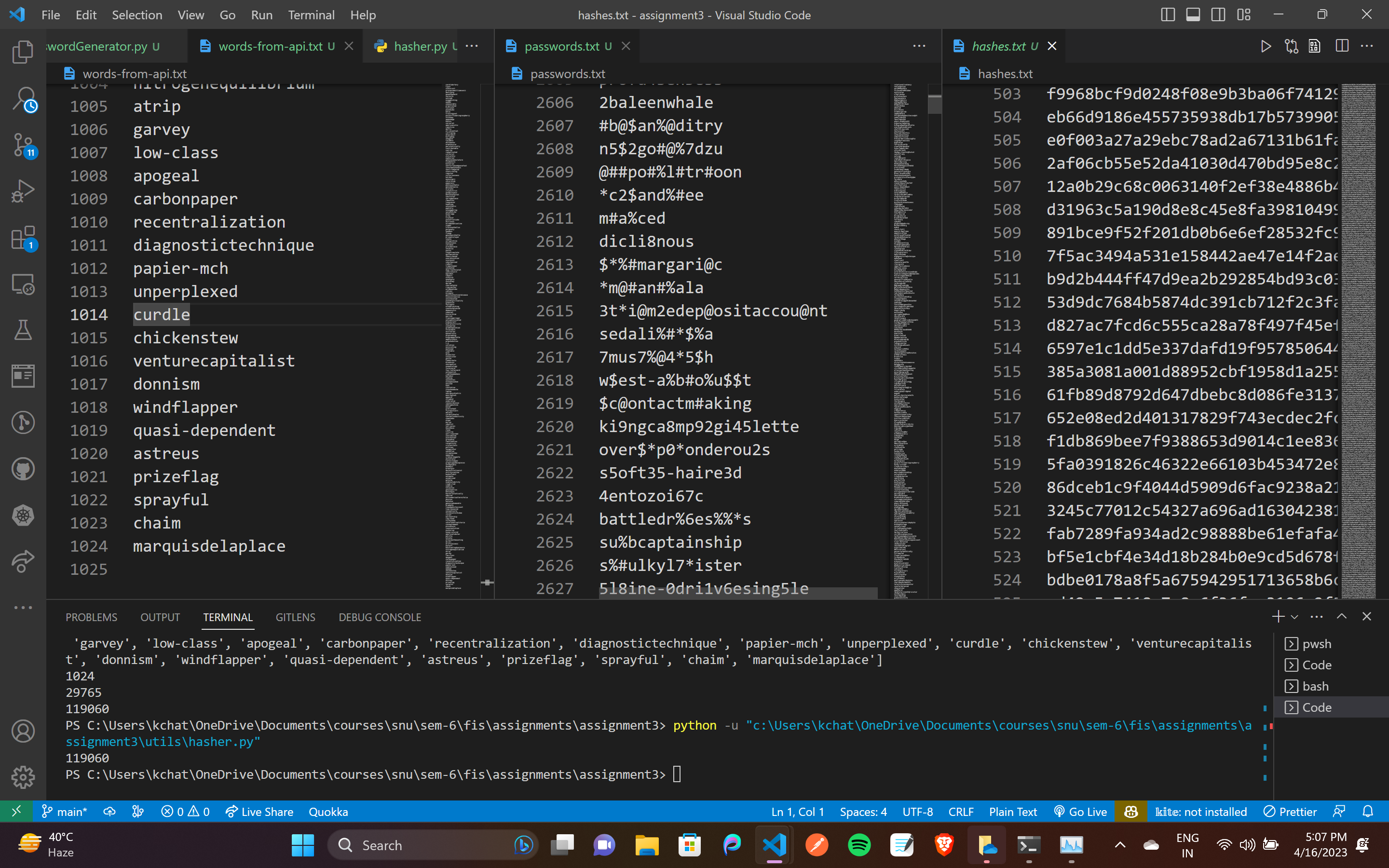
This function invokes the randomization process and implements the multiplier. Finally, it writes the passwords to the file.

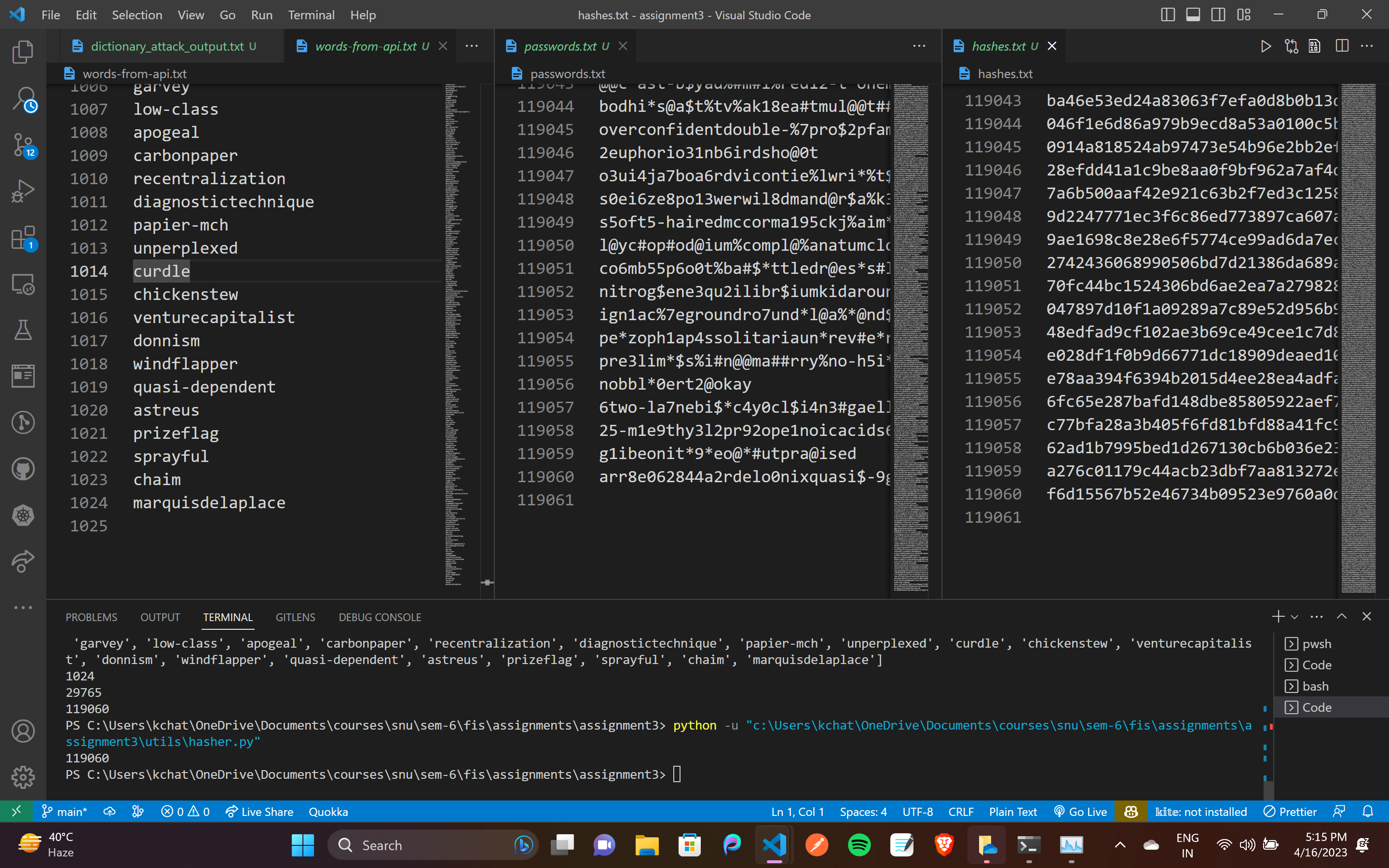
**I generated 119060 Passwords.**

** 1024 words were fetched from the API. There were 29765 passwords after randomization (adding numbers and symbols) and 119060 passwords after multiplying.**

**Step 3: Hashing the passwords**

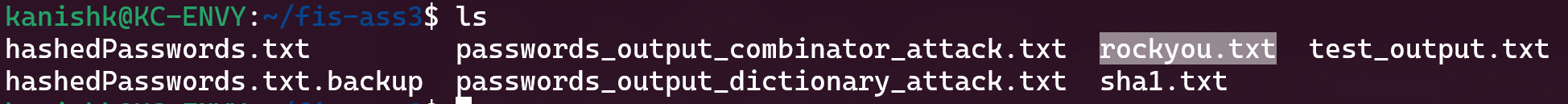
Done using hashlib module in python. The strings were encoded and then hashed. Output file hashedPasswords.txt was generated.





**Step 4: Getting a wordlist/dictionary**

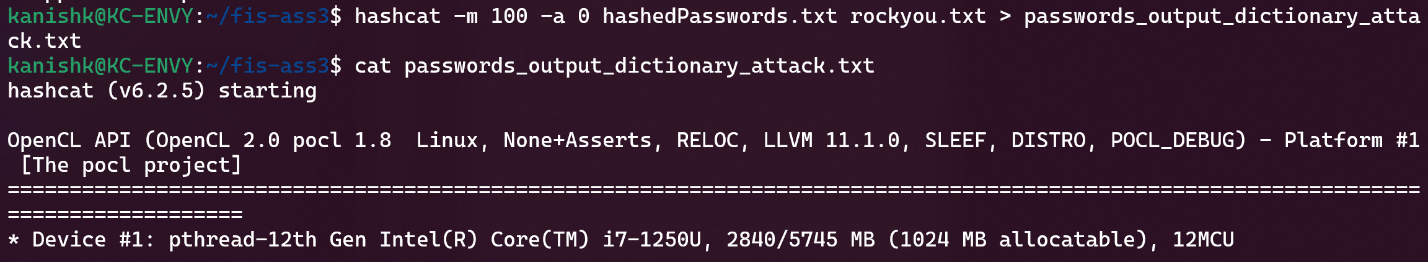
In order to perform a hashcat attack, we need a dictionary of commonly used passwords. I went with rockyou.txt, which is a very common passwords dictionary and comes pre-installed by default in Kali linux.



**Step 5: Performing Dictionary Attack using Hashcat**

**-m 100** means SHA1 **-a 0** means dictionary attack

I stored the commands output in a text file.

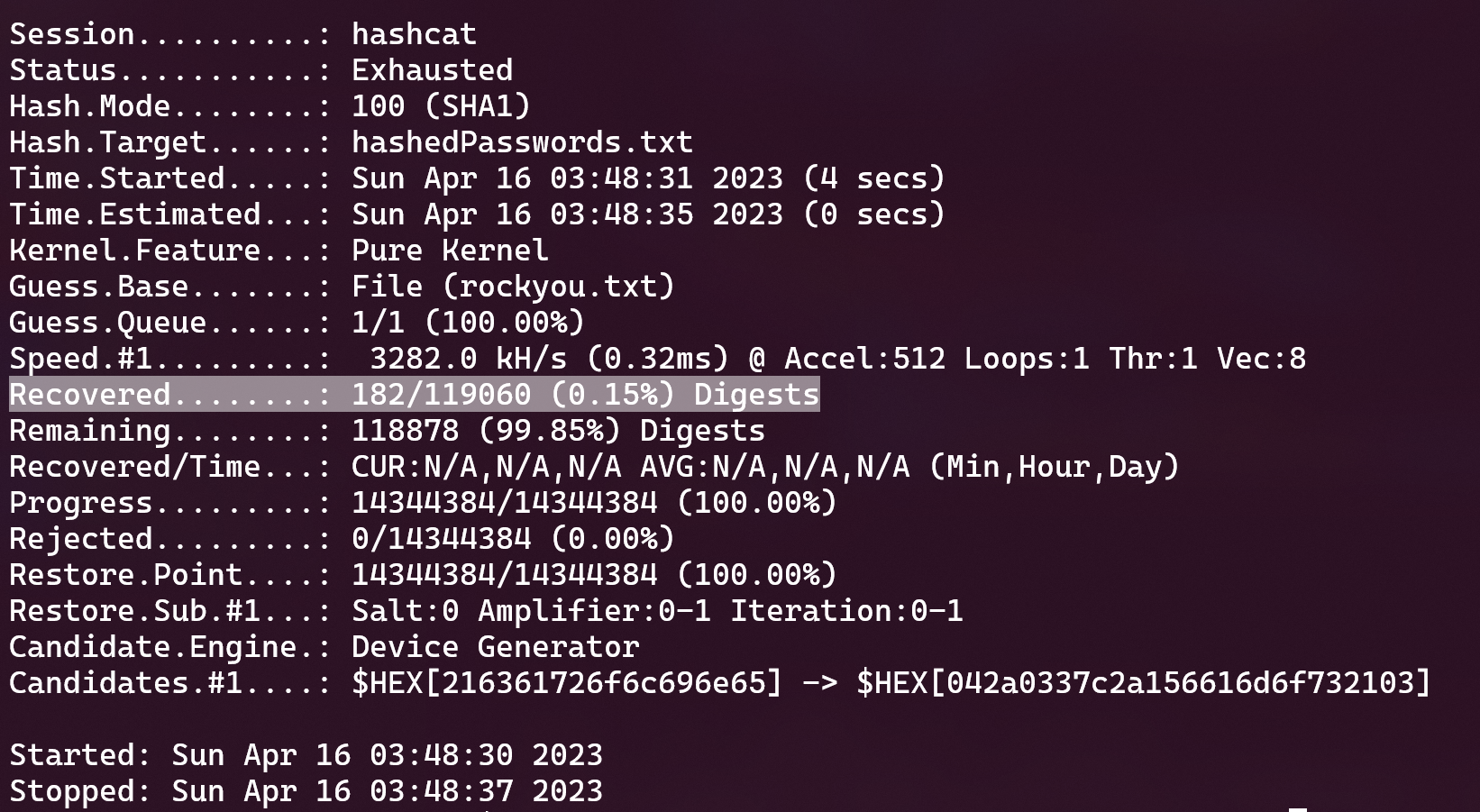
****

**Step 6: (BONUS) Performing Combinator Attack using Hashcat**

I tried a combinator attack, but further research found it would take weeks(!!) so I had to cancel it.

**RESULT**

**Hashcat was run on Ubuntu running on WSL in an ultrabook powered by Intel Core i7-1250U.**

****

It recovered 182/119060 hashes which translates to 0.15%.

This incredibly low recovery rate is due to the fact that the passwords generated were very complex after going through the iterative randomization and multiplication process.