

<b>Course:</b>	<b>INFO3142</b>
<b>Professor:</b>	<b>Jim Cooper</b>
<b>Project:</b>	<b>Lab #1</b>
<b>Due Date:</b>	<b>Wednesday, Sep 17, 11:30 pm</b>
<b>Submitting:</b>	<b>Submit your zipped solution to the Lab 1 submission folder</b>

## How will my lab be marked?

<b>Marks Available</b>	<b>What are the Marks Awarded For?</b>	<b>Mark Assigned</b>
1	Loading of the four index files provided into a suitable Python data structure (list or dictionary) as specified in project requirements	
1	Creation of data entry loop which processes each new anagram (word) and terminates if a blank entry is received. Convert entered words to upper case	
1	Implementation of “permutations” function which receives a new anagram (word) and returns a unique list of all possible letter permutations.	
1	Uses the permutations list to search the word dictionary (or list) for any and all matches and displays each matched word with the corresponding part of speech (noun, verb, adjective, adverb)	
1	Implementation of timing as “hh:mm:ss” format for each new word as it’s tested (see sample output for placement)	
5	Total	

## Lab Description

Create a single Python program using the editor/IDE of your choice. Name this source file using the following template: your first name followed by an underscore and then “Lab1”. For example, “Jim\_Lab1.py”.

The main goal of this lab is to use the four modified WordNet dictionary index files and some Python code to build an application that can solve anagram word game puzzles.

Although it’s not needed for the lab, here’s the link to the WordNet project at Princeton:

<https://wordnet.princeton.edu/>

The user enters a series of letters such as “BEALF” and your code will return: “fable” “noun”

...where “fable” is the decoded English word and “noun” is the part of speech. See the example output later in this PDF file.

**Note:** You'll find many "Jumble Puzzle" code solutions on the web in various programming languages; however, in this case, we want you to use the data files provided and write your own code for most parts of the project.

One of the main challenges with this project is generating all the possible permutations for a given set of characters. For example, the anagram "BEALF" has 120 different ways of combining the characters. Web sites such as Stack Overflow have multiple versions of "permutations" functions and you may use one of these code fragments in your solution so long as you include all the source code as a function in your code. Find an example which takes a single test "word" such as "BEALF" as an input parameter and returns a Python list of all possible unique permutations. You may also use an LLM to create a Python function that does the job and returns the permutations list.

You can create your own "permutations" function for the learning experience; however I would recommend doing that at the end because it will make debugging easier to start with something you know works.

The only role played by the WordNet index data in this case is to tell us if one of the permutations is an English word. It's also possible that any given anagram could contain more than one matching English word.

In some cases, you won't find a match even though the word exists. For example, the "SKURNH" anagram is intended to be "SHRUNK" however WordNet only has "shrunk"

## JUMBLE

Unscramble these Jumbles,  
one letter to each square,  
to form four ordinary words.

BEALF

FROEF

CUVAMU

SKURNH

©2020 Tribune Content Agency, LLC  
All Rights Reserved.

**Answer " here:** "

### THAT SCRAMBLED WORD GAME

By David L. Hoyt and Jeff Knurek

Get the free JUST JUMBLE app • Follow us on Twitter @PlayJumble

10/16

THE TOWN'S LANDFILL WAS OVER CAPACITY AND BEGINNING TO ---

Now arrange the circled letters to form the surprise answer, as suggested by the above cartoon.

## Detailed Steps for Doing Lab #1

You can create a solution to the jumble puzzle problem using either a Python nested list or a Python dictionary. It's recommended that you use a dictionary because the performance will be better.

Begin by loading the following index files into your dictionary or nested list:

NounsIndex.txt  
VerbsIndex.txt  
AdjIndex.txt  
AdvIndex.txt

The "key" for each entry in the dictionary should be an English word from one of the index files. You will not need to load any multi-level n-grams as the Jumble Puzzle game only uses single words.

The "value" property for each dictionary element should be the correct part of speech for the word, i.e., noun, verb, adjective, adverb.

The above is only a suggestion on how to manage the data for the lab, however **it's a requirement that you use the WordNet derived index files as provided.**

Because words that are longer than five or six characters have many more possible permutations, we want to time our code to see how long the processing takes. See the output below for a demonstration of how this should look and also see the “Notes” section for some help getting and formatting the current time. Larger anagrams take longer.

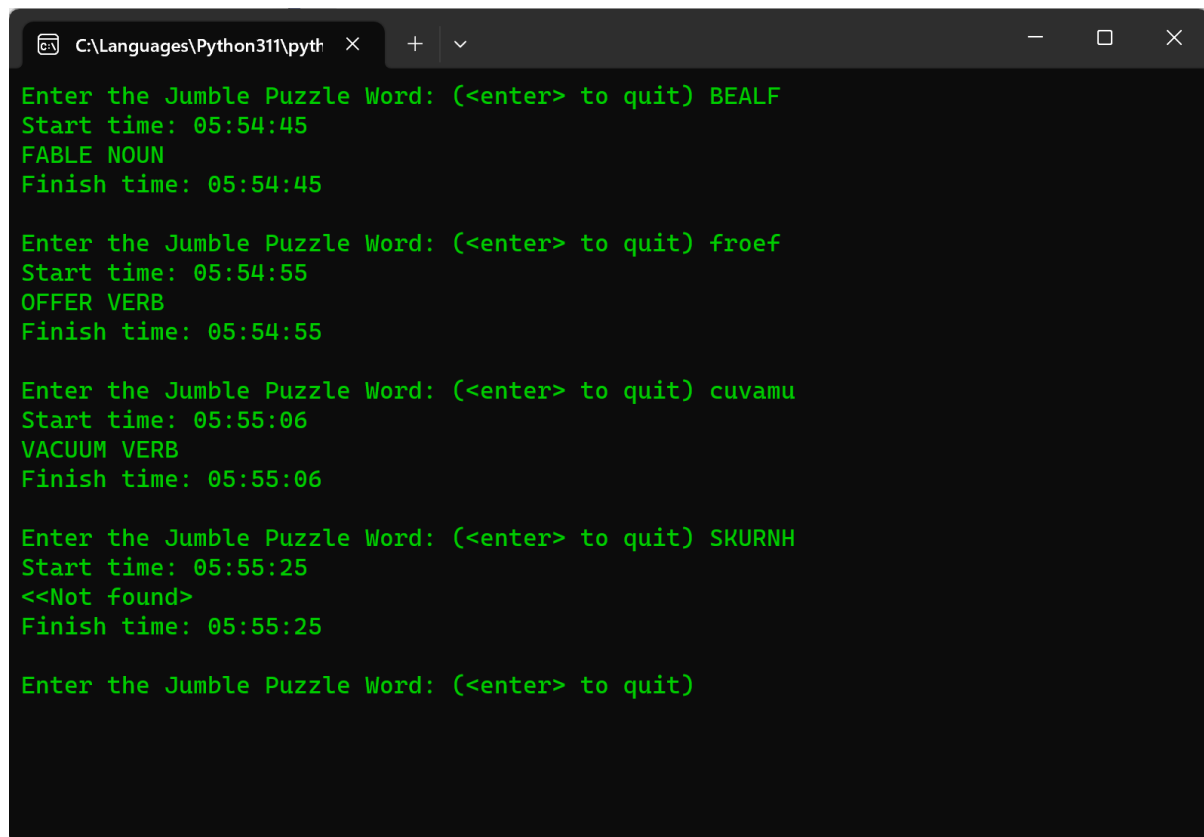
Finally, creating a solution for this project is best done by starting with the sample output and doing some research to determine what it takes to create the code to do the same thing.

## Notes:

Here’s some sample code for getting the current time:

```
import time
curr_time = time.strftime("%H:%M:%S", time.localtime())
```

## Sample Output:



```
C:\Languages\Python311\pyth x + v
Enter the Jumble Puzzle Word: (<enter> to quit) BEALF
Start time: 05:54:45
FABLE NOUN
Finish time: 05:54:45

Enter the Jumble Puzzle Word: (<enter> to quit) froef
Start time: 05:54:55
OFFER VERB
Finish time: 05:54:55

Enter the Jumble Puzzle Word: (<enter> to quit) cuvamu
Start time: 05:55:06
VACUUM VERB
Finish time: 05:55:06

Enter the Jumble Puzzle Word: (<enter> to quit) SKURNH
Start time: 05:55:25
<<Not found>
Finish time: 05:55:25

Enter the Jumble Puzzle Word: (<enter> to quit)
```