Homework 0 for CSI 431

Due: 9/10/2020 (This assignment will not be graded, we will upload solutions on 9/10)

Instructions:

This assignment is made of only programming problems. Its goal is to build your understanding of python, but also help you learn some advanced language features that might come in handy later on. Use the documentation for the last stable version to read up on things that are new to you: https://docs.python.org/3.6/

You do not have to submit your assignments, but it is recommended to organize your code into **one hw0.py python file** and also save **one** results.txt **text file** to be able to compare your solution to the one we will upload. Below is a sequence of subproblems (solve in this order)

1- **Install and Test Python Environment** Install python3 environment, save the following code with name "hw0.py" (note that python is indentation sensitive), and run it in the terminal with "python hw0.py" or in your preferred IDE.

```
def test_print():
    print("This is a test statement.")
if __name__ == '__main__':
    test_print()
```

2- **Define Objects** Define a function called 'list_set_length' to assign items_list= [1, 2, 3, 4, 3, 2, 1] as a list object, and items_set = {1, 2, 3, 4, 3, 2, 1} as a set object. Use the len function, print the size of the list and size of the set.

3- **Comprehension** Python provides for expressions called comprehensions that let you build collections out of other collections. Here's an example:

```
>>>{x*y for x in {1,2,3} for y in {2,3,4}}
{2, 3, 4, 6, 8, 9, 12}
```

This is said to be a set comprehension because its values are sets. The notation is similar to the traditional mathematical notation for expressing sets in terms of other sets. To compute the value, Python iterates over the elements of the sets, temporarily binding the control variables x and y to each element in turn and evaluating the expression x*y in the context of that binding. Each of the values obtained is an element of the final set.
Assume that S and T are assigned sets. Without using the intersection operator (&), write a function called 'set_intersect' that uses comprehension to returns the intersection of S and T. Hint: Use a membership test in a conditional statement at the end of the comprehension. Try out your comprehension with S = {1,2,3,4} and T = {3,4,5,6}.

4- **Tuples:** Suppose S is a set of integers, e.g. {−4,−2, 1, 2, 5, 0}. Write a function 'three_tuples' that includes a triple comprehension and returns a *list* of all three-element *tuples* (i, j, k) such that i, j, k are elements of S whose sum is zero. Note it is possible that i=j=k (e.g. (0,0,0) is in the solution set).

5- **Dictionaries:** Conceptually, a dictionary is a set of key-value pairs. The syntax for specifying a dictionary in terms of its key-value pairs resembles the syntax for sets—it uses curly braces—except that instead of listing the elements of the set, one lists the key-value pairs. In this syntax, each key-value pair is written using colon notation:

```
key : value
```

   a) Write a function 'dict_init' to initialize the following dictionary:

```
mydict = {'Neo':'Keanu', 'Morpheus':'Laurence', 'Trinity':'Carrie-Anne'}
```

   b) Suppose dlist is a list of dictionaries and k is a key. Write a function 'dict_find' that receives dlist and k, and uses a comprehension to return a list whose i^{th} element is the value corresponding to key k in the i^{th} dictionary in dlist. If a dictionary does not contain k as one of its keys, use 'NOT PRESENT' for that dictionary.

6- **File reading:** Write a function 'file_line_count' to open a file and return the number of lines that it has. Try out your function on stories_small.txt.

7- **Mini Search Engine:** Given a file of "documents" where each document occupies a line of the file, you are to build a data structure (called an inverse index) that allows you to identify those documents containing a given word. We will identify the documents by document number: the document represented by the first line of the file is document number 0, that represented by the second line is document number 1, and so on. Make matching of case-insensitive (e.g. Wall and wall are the same words)

   Note that the period is considered part of a substring. To make this easier, we have a file of documents in which punctuation are separated from words by spaces. Often one wants to iterate through the elements of a list while keeping track of the indices of the elements. Python provides enumerate(L) for this purpose.

   a) Write a procedure make_inverse_index(strlist) that, given a list of strings (documents), returns a dictionary that maps each word to the set consisting of the document numbers of documents in which that word appears. This dictionary is called an inverse index. (Hint: use enumerate.)
   b) Write a procedure or_search(inverseIndex, query) which takes an inverse index and a list of words query, and returns the set of document numbers specifying all documents that contain any of the words in query.
   c) Write a procedure and_search(inverseIndex, query) which takes an inverse index and a list of words query, and returns the set of document numbers specifying all documents that contain all of the words in query.
   d) Try out your procedures on stories.txt. Try *or_search, and_search* with the queries "united states" and "wall street" on the documents in stories.txt and save the resulting lists (tw lists of document ids per query) in a file named results.txt.