CS 121 – Lab #8

It would be a good idea to practice two concepts you have seen recently: Python exceptions, and dictionaries. Create a new folder on your USB drive or account called lab08, and do all your work for this lab in that folder. From the class Web site, please download the file summer.txt. This is an input file you will use today.

There are three parts to the lab. You may implement all parts in the same source file. Since we will be practicing with several exceptions, you could call the program except.py. Or you may do each part in a different source file. Open Anaconda Spyder as usual. You may find it helpful to refer to your notes on exceptions and dictionaries. On the class Web site, they can be found in the files file.pdf and dict.pdf.

Part 1: ValueError

Let's ask the user to enter a *positive integer*, and then print out the square of this number. Sounds easy enough. But let's also add error checking: The user might not comply with your instructions! For example, the I/O might proceed like this:

```
Please enter a positive integer:  two
Sorry, I don't understand your input.  Try again.

Please enter a positive integer:  2.5
Sorry, I don't understand your input.  Try again.

Please enter a positive integer:  -2
Your number is not positive.  Try again.

Please enter a positive integer:  2
Your number squared is 4
```

You should have a `while` loop that contains the code performing the input. And you should also use `try`/`except` to prepare for the possibility that the user's input cannot be converted to an integer. You should also check that the number is greater than zero.

Part 2: ValueError while tokenizing

Ask the user to enter a line of text, consisting of both words and numbers. For example, the user might enter "3 ducks, 4 chickens, and 5 geese." We want to add up all the (real) numbers that appear in the line. Ignore the words. Here is a general outline of the solution:

Tokenize the line, creating a list of strings called `tokenlist`.
total = 0.0

For each token in the tokenlist,
  if the value can be converted to a real number,
    increment the total by that value
  but if the value can't be converted to a real number,
    then skip it
Print the total.

Your program should not print an error message when it sees a string that is not a number. It should only print the total. If the string had no numbers, then the sum is zero.

Part 3: Read a file to create a dictionary

Let's implement an idea we saw in class recently: storing a dictionary of Olympic host cities. You have a file called summer.txt, which will be the input file for this part. You should create a dictionary variable called `host`. In this dictionary, the key will be the year number, and the corresponding value will be the name of the host city. Inside summer.txt, the first 4 characters on each line give the year number. The rest of the line, line[5 : –1], contains the name of the host city.

First, your program should ask the user to enter the name of the input file. The user will probably type in summer.txt, but maybe someday we'll also see winter.txt.

Your program needs to check that the file actually exists. Use try/except, and give the user another chance in case the file name was entered incorrectly. You will need to use a while loop. What is the name of the exception that occurs when we can't open a file?

Next, after your program has opened the file successfully, you should read it line by line. Each line contains a year and a host city name. This key-value pair will be an entry in the `host` dictionary. Don't forget to close the input file when you are done reading it.

At this point, your dictionary has been created, so now let's let the user query the dictionary. Ask the user to enter a year, such as 1956. The program should use the dictionary to look up the host city for the Olympics that year. However, the user might enter a year, such as 1957, when there was no Olympics. Rather than using try/except, you can use the Python word "in" to first inquire whether the key exists in the dictionary. Print a suitable error message such as "I have no data on that year." You don't need to enclose this interactive I/O in a loop.

The I/O should look something like this:

```
Which file contains the Olympic host data?  spring.txt
Sorry, I can't find that file.  Try again.

Which file contains the Olympic host data?  summer.txt

Enter a year:  1960
The host in 1960 was Rome.
```