Download the folder **lab\_7\_starting.zip**, unzip it and open it as a PyCharm project. It contains several **.py** and **.txt** files for this and later Labs and Homework. Do as many of the following as you can, submitting each to the **L7 Assignment** link on Canvas.

**[L7-1]** (**count\_vowels.py**) Complete the function **count\_vowels(str2count)**. It should return the number of lower-case vowels (any one of**'a','e','i','o','u'**) within **str2count**. The provided **main()** reads a string from the user, and prints out the value returned by calling the function.

**[L7-2]** (**build\_list.py**) Write a program that reads **int N** then builds a list **num\_list** such that **num\_list[i] = i** for **i == 0..N-1.** Do this by using the accumulator pattern: initialize **num\_list** to the empty list **[]**, then iterate over each integer in the range, appending it (**num\_list.append(...)**) to the end of **num\_list**. At the end, print out your final value of **num\_list**.

**[L7-3]** (**build\_list2.py)** Same as **[L7-2]** but build **num\_list** by concatenating the **list[num]** to the end of **numlist** (**numlist = numlist + [list[num]]**), then assigning the result to **num\_list**. Print out the final value of **num\_list** as before.

**[L7-4]** (**build\_list3.py**) Same as **[L7-2]**, but build **num\_list** by using a list comprehension: **num\_list = [num for num in range(21)]**.

**Lab Exploration:** Which is the fastest of the three above approaches? We'll use IPython's timing functions within a Jupyter notebook to explore this...

**[L7-5]** (**remove\_punct.py**) Complete the function **del\_punct(st)**. It should return **st** with all of its punctuation characters removed. Define a punctuation character to be any character found in the imported **string** module **string.punctuation**.

**[L7-6]** (**print\_words.py**) The starting code opens and reads the text file **words\_2.txt** into a single string. Unlike **words.txt**, this file contains words with embedded punctuation (single quotes (**'**) and forward slashes (**/**)).

Do each of the following by completing the starting functions (names given) to compute and print the requested information. The first function will return a list of all words, which is then passed as an argument to the remaining functions for computing and printing. Note the advantages of returning values from a function instead of printing them out directly within it; this increases the reusability of our code, as well as allowing us to write tests for these functions.

(a) (**def split\_into\_words(bstring)**) Split the single string into a list **all\_words**. Then print out its length and return the entire list. Use **str**'s **split()** method with no arguments.

(b) (**def print\_longest\_v1(all\_words)**) Iterate over each element **all\_words**, finding and printing the longest word; if more than one word has the same length, print any one of them. Hint: use an accumulator pattern, tracking both longest word so far and length of such a word. Better: try to print out all longest words (those tying for longest length), by accumulating a list of longest-so-far words.

(c) (**def print\_longest\_v2(all\_words)**) Do (b) another way: build a list **len\_list** containing tuples **(len(word),word)** for each word, sort it using **list**'s **sort()** method, then print out the last element.

(d) (**def print\_first\_2\_equals\_last\_2(all\_words)**) Print out all words whose first **2** characters are the same as the last **2**. Hint: use slices and be careful about the lengths of strings.