Laboratory 9: Dictionaries

1. Overview

In this laboratory, we create an index of the long words in a text file. Each word is associated with

2. Details

A Python dictionary is a data-structure that associates keys with values. The keys can be almost anything, such as strings or numbers, but they have to be immutable. The values can be literally anything. A Python dictionary is indicated by braces ({ }) that contain key-value pairs where the key is separated from the value by a colon (:). To define a dictionary, we can write the key-value pairs inside a dictionary as in

```
      dicti = {"one":1, "two":2, "three":3, "four":4}
```

or by using list notation to add new elements, as in

```
      dicti["five"]=5
      dicti['six']=6
```

Normally, we start out with an empty dictionary and then add values using the list notation.

Task 1: Write a function `process` that takes as input a filename, opens the corresponding file, and reads the file line by line. You can use the provided file 'alice.txt'. The file then splits the line along white spaces to obtain a list of words. Each word itself is stripped of leading and trailing punctuation marks, including the underscore, the hyphen, etc. The function then prints out all the words of length larger than 8. You will probably have to add to your lists of letters to be stripped from the word.

If you look closely, you will find that the 'alice.txt' file has some weird pseudo-words, where there are hyphens in the interior of a word. This is an artifact of the transliteration and we need to exclude those words from consideration.

Task 2: Restrict the function to only print out words that do not contain an interior hyphen.

Now that we can isolate all long word from the text, we need to be able to locate them. We do so by keeping track of line numbers.

Task 3: Add a variable line-number to your function that is incremented whenever we find a new line. Now print out the word together with the line-number of the line in the word.

Our next step is to create a dictionary that associates a long-word with a **list of line-numbers.** At first, the dictionary is empty. Whenever we have a new word to enter in the list, we add a key-value pair to the dictionary. The key is the word, then value is a list with only the line-number as an element. If the word is already in the dictionary, then we append the current line-number to the value. How do we know whether a word is already in the dictionary? We know because we can ask:

```
      if word in index:
```

where index is the name of the dictionary.

Task 4: Add the dictionary ‘index’ to the function and return the dictionary.

As you can see, the dictionary is not easy to look at. What we will do is to sort the keys in the dictionary, and then print out the dictionary word for sorted word. To obtain all the keys in the dictionary ‘index’, you just say `index.keys()`. The result is a list with all the words that serve as keys. This list can be sorted. Now, we go through the sorted list of keys and print out the corresponding values. For this we use the `index[word]` notation that returns the value of the key `word` of a dictionary `index`.

Task 5: Write a function pretty-print that takes as sole variable a dictionary. The function then prints out on separate lines all the keys in order followed by the value.

If you combine these two functions, then the result for ‘alice.txt’ should be starting out with

```
accusation [1146]
accustomed [332]
addressed [642]
adventures [15, 38, 1301]
advisable [338, 350]
affectionately [616, 1077]
afterwards [1274]
alternately [825]
altogether [232]
anxiously [174, 496, 993, 1023, 1197]
appearance [1052]
archbishop [337, 347]
assembled [312, 1132]
attending [460]
authority [323]
barrowful [600, 602]
…
```

3. Overview

Dictionaries can also be used as counters. In fact, there is a Counter in the built-in collections class that you can use as an alternative in order to do the following exercises. We will now write a function that calculates the number of letters and letter combinations in a DNA file, 2021.txt.

4. Details

Task 6: Familiarize yourself with the file. Then write a function that will print out all lines that contain a letter other than ‘A’, ‘C’, ‘G’, and ’T’. Of course, you need to strip the newline character from each line.

Task 7: Create a dictionary called counter that is initially

```
      dicti = {'A': 0, 'C': 0, 'G' : 0, 'T': 0, 'N': 0}
```

For every letter in the alphabet, increment the counter value when you see it. Then print out in a nice format the total number for each letter.

Task 8: As you see, the letters are not equally distributed. What about pairs of letters? Create a counter for all pairs of letters. Notice, that an occurrence of a pair can be distributed over two lines. This means that we need to handle the beginning and the end of a line differently. In particular, we need to "remember" the last letter of a line when we start processing the next line.