Unit Project 1: The Indexer

Introduction

The unit projects are designed for all students to get experience with building a complex system from the ground up, using the core concepts of Object Oriented Programming (OOP) as well as all the Python programming tools at their disposal.

The final chat system will have the ability to return a sonnet (to a lone user) as well as retrieve past dialogue (a chat history) using keywords. UP1 is about implementing these two pieces of functionality.

Important files for UP1:

- ***indexer_student.py*: You will need to modify this file (and this file only)**
- *AllSonnets.txt*: A small collection of Shakespeare's sonnets
- *roman.txt*: provide the integers to roman numerals conversions
- *roman2num.py*: A Python module for converting integers to roman numerals
- *roman.txt.pk*: A binary file which contains a roman number - integer mappings
- *p1.txt*: the first sonnet, just for testing purposes

You will implement functions for ***two*** classes: the base class `Index`, and a derived class `PIndex` which **inherits** data members and functions from `Index`. We will explain the requirements for each of them in the following pages.

The Base (Super) Class: `Index`

```
 9 class Index:
10     def __init__(self, name):
11         self.name = name
12         self.msgs = [];
13         self.index = {}
14         self.total_msgs = 0
15         self.total_words = 0
16 
17     def get_total_words(self):
18         return self.total_words
19 
20     def get_msg_size(self):
21         return self.total_msgs
22 
23     def get_msg(self, n):
24         return self.msgs[n]
25 
26     # implement
27     def add_msg(self, m):
28         return
29 
30     def add_msg_and_index(self, m):
31         self.add_msg(m)
32         line_at = self.total_msgs - 1
33         self.indexing(m, line_at)
34 
35     # implement
36     def indexing(self, m, l):
37         return
```

Given above is the first segment of the code you will work on. This idea is for an object of this class to store a message (via the `add_msg` function), and then store a message *and* index the words (`add_msg_and_index`). The messages are stored in the data member `self.msgs`, which is a list of strings. It stores them according to the receiving order, so the *i*-th message is stored in `msgs[i]`. You need to implement `add_msg`.

The member `self.index`, a dictionary, maps a word (e.g. “thy”) to a list. Each item of the list refers to the line number of the message in which the word appears. You need to implement the `indexing` function for this to work.

The main query interface is, unsurprisingly, the member function `search`.

```
    # implement: query interface
'''
return a list of tupple. if index the first sonnet (p1.txt), then
call this function with term 'thy' will return the following:
[(7, " Feed'st thy light's flame with self-substantial fuel,"),
 (9, ' Thy self thy foe, to thy sweet self too cruel:'),
 (9, ' Thy self thy foe, to thy sweet self too cruel:'),
 (12, ' Within thine own bud buriest thy content,')]

'''
    def search(self, term):
        msgs = []
        return msgs
```

After implementing the `Index` class, test it by feeding arbitrary text to it. For example:

```
In [59]: import indexer

In [60]: my_idx = Index('hahaha')

In [61]: my_idx.add_msg_and_index("what is ths thing called?")

In [62]: my_idx.add_msg_and_index("who knows who?")

In [63]: my_idx.search('who')
Out[63]: [(1, 'who knows who?')]
```

“who” and “who?” are treated as separate words. This is a bug! (But don’t deal with it just yet)

The Derived (Sub) Class: `PIndex`

We want to index the entire collection of sonnets. Using the `Index` class, this can be done by reading lines off the AllSonnets.txt, and creating an index. We can do this with just the console:

```
In [52]: import indexer

In [53]: idx = Index("test")

In [54]: lines = open("AllSonnets.txt", 'r').readlines()

In [55]: for l in lines:
    ...:     idx.add_msg_and_index(l)
    ...:

In [56]: idx.search('love')
Out[56]:
[(169, ' No love toward others in that bosom sits\n'),
 (176, "For shame deny that thou bear'st love to any,\n"),
 (188, ' Make thee another self for love of me,\n'),
 (283, ' And all in war with Time for love of you,\n'),
 (360, ' My love shall in my verse ever live young.\n'),
 (379, " Mine be thy love and thy love's use their treasure.\n"),
 (394, ' And then believe me, my love is as fair\n'),
```

But we have one ***additional*** requirement; we want to retrieve an arbitrary poem from the collection at will! Here is how things will look at the console:

```
In [57]: sonnets = PIndex("AllSonnets.txt")

In [58]: sonnets.get_poem(1)
Out[58]:
['I.',
 '',
 'From fairest creatures we desire increase,',
 " That thereby beauty's rose might never die,",
 ' But as the riper should by time decease,',
 ' His tender heir might bear his memory:',
 ' But thou contracted to thine own bright eyes,',
 " Feed'st thy light's flame with self-substantial fuel,",
 ' Making a famine where abundance lies,',
 ' Thy self thy foe, to thy sweet self too cruel:',
 " Thou that art now the world's fresh ornament,",
 ' And only herald to the gaudy spring,',
 ' Within thine own bud buriest thy content,',
 " And, tender churl, mak'st waste in niggarding:",
 ' Pity the world, or else this glutton be,',
 " To eat the world's due, by the grave and thee.",
 '',
 '',
 '']
```

But, of course, we will want to search through our poetry as well!

```
In [50]: sonnets.search('five')
Out[50]:
[(1112, ' Even of five hundred courses of the sun,'),
 (2674, ' But my five wits nor my five senses can'),
 (2674, ' But my five wits nor my five senses can')]
```

So, we want to *inherit* the class `Index`, and then add a few new member functions:

```
53 class PIndex(Index):
54     def __init__(self, name):
55         super().__init__(name)
56         roman_int_f = open('roman.txt.pk', 'rb')
57         self.int2roman = pickle.load(roman_int_f)
58         roman_int_f.close()
59         self.load_poems()
60
61         # implement: 1) open the file for read, then call
62         # the base class's add_msg_and_index
63     def load_poems(self):
64        return
65
66         # implement: p is an integer, get_poem(1) returns a list,
67         # each item is one line of the 1st sonnet
68     def get_poem(self, p):
69         poem = []
70         return poem
```

The member function `load_poems` opens the file (name stored in `name`), and feeds the lines to the base class' method `add_msg_and_index`

The trickier thing is to retrieve, say, poem #3. The basic logic is:

- convert 3 to its roman numeral string "III"
- Pad ".", making it "III."
- Use the above the search, finding the starting line
- Retrieving lines until you hit the beginning of the next sonnet, or the end.

We have supplied a converting code, in roman2int.py. It reads a table, and generates two dictionaries, for for roman to int, and another int to roman. In the picture above, line 57 does that.

More Improvements

Once you complete the above requirements, play around and see what you can improve. There are a few annoying things:

- “hey.” is treated as word (not “hey”) to be indexed. Basically the trailing punctuation marks are not removed. Be careful, if you remove them, then the logic of finding the heading of a poem can be affected, because (“I”, “X”) can be both roman numbers, and as words.

- It does not detect duplicates. Each appearance of a word, even in the same sentence, counts separately. How do you remove them? As in:

```
In [50]: sonnets.search('five')
Out[50]:
[(1112, ' Even of five hundred courses of the sun,'),
 (2674, ' But my five wits nor my five senses can'),
 (2674, ' But my five wits nor my five senses can')]
```

- How would you implement a search for phrases, e.g. “five hundred”?

- And many others!

TL;DR - Summary: UP1 Specifications

You need to implement the missing functionality of Index and PIndex classes.

Index:

- add_msg(m) - Appends string m to self.msgs and increments self.total_msgs
- indexing(m,l) - Splits a string m (located at line number l) into individual words and then updates the dictionary self.index which is a mapping from words to their frequencies
- search(term) - Returns a list of tuples which specify each line number (and line) in which term appears

PIndex:

- load_poems() - Opens the file given by self.name for reading and then indexes every line, using the base class implementation of add_msg_and_index()
- get_poem(p) - Returns poem p as a list of strings