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.

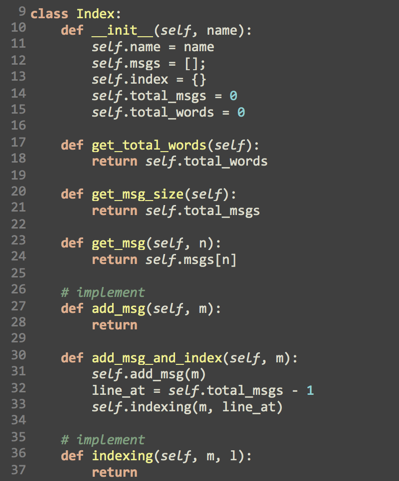
The final chat system will have the ability to return a sonnet (to a lone user) as well as retrieve past dialogue (a chat history) using keywords. UP1 is about implementing these two pieces of functionality.

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



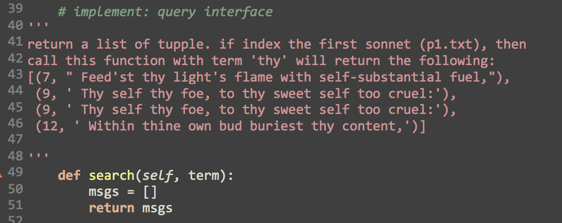
Given above is the first segment of the code you will work on. This idea is for an object of this class to store a message (via the get\_msg function), and then store a message *and* index the words (get\_msg\_and\_index). The messages are stored in the data member self.msgs, which is a list of strings. 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. [JF]: in the code, def indexing(self, m, l):, “l” means line number? Which should be related to self.total\_msgs?

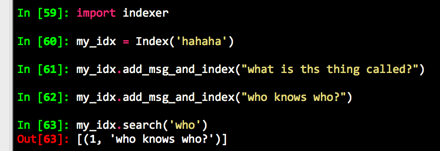
The main query interface is, unsurprisingly, the member function search.

[JF]:return a list of tuples, each tuple has 2 elements, 1st is line number, 2nd is the message.

[JF]: no need to have msgs=[] in the given code?



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



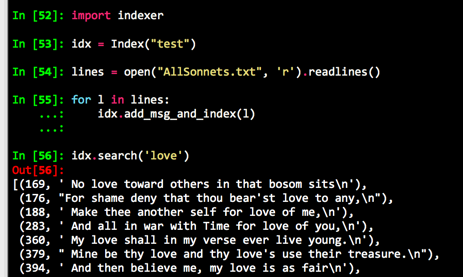
“who” and “who?” are treated as separate words. This is a bug! (But don’t deal with it just yet)

# 

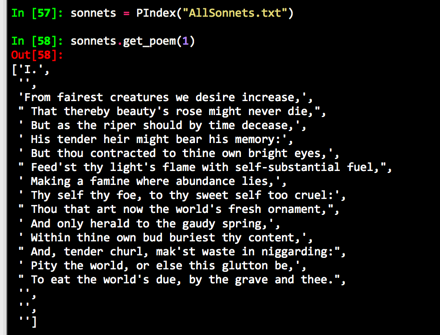
# 

# The Derived (Sub) Class: PIndex

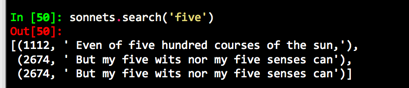
We want to index the entire collection of sonnets. Using the Index class, this can be done by reading lines off the AllSonnets.txt, and creating an index. We can do this with just the console:



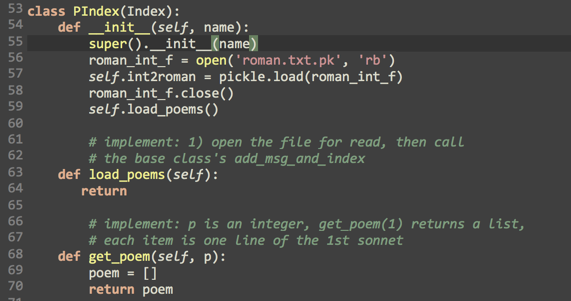
But we have one ***additional*** requirement; we want to retrieve an arbitrary poem from the collection at will! Here is how things will look at the console:



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



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



The member function load\_poems opens the file (name stored in name), and feeds the lines to the base class’ method get\_msg\_and\_index 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.

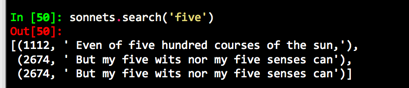
[(go\_line, m)] = self.search(p\_str), there could be multiple lines with the same poem number?

Or will select the lines that is not a poem number but actually a poem line?

# 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:



* 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