Programming Problem:
Finding the principal characters of a literary text

MARCH 27, 2015

Programming Problem

Write a program that reads a literary text (e.g., “War and Peace” or “The Illiad”) and does simple text analysis to figure out the *principal characters* of the novel.

For example, when I ran my program on “The Illiad” the most frequent characters were:

(563, 'Trojans'), (548, 'Achaeans'), (447, 'Jove'),
(421, 'Hector'), (383, 'Achilles'), (183, 'Agamemnon'),
(178, 'Priam'), (160, 'Patroclus'), (146, 'Minerva'), (137, 'Ajax')

Motivation

Solving this problem will provide additional illustration of new Python features we have learned:

- Slices of lists and strings
- String operations (e.g., split, join, etc.)
- List comprehensions

Main Idea

- Since character names are proper nouns, starting with upper case letters, the idea is to look for words starting with upper case letters that do not appear at the beginning of sentences.
- So the program will partition the text into sentences, assuming that ".", "!", and "?" are all the possible sentence delimiters.
- Then it counts the frequency of the proper nouns and reports the most frequent of these. We only keep names that are at least 4 letters long.

Function parseSentences

```
# Takes a string as parameter and "splits" it into "sentences."
# We assume that ".", "!", and "?" are sentence delimiters

def parseSentences(bigString):
    return bigString.replace("!", ".").replace("?", ".").split(".")
```

- bigString is indeed a big string, representing the entire file.
- This function returns a list – each element in the list is a string representing a sentence.

Next task: split sentences into word sequences

- We have solved this problem earlier and written a function called “parse” for it.
- That algorithm examined the string character-by-character and pulled out contiguous sequences of letters.
- Now we will use a different algorithm to solve this problem.

Algorithmic Idea

1. Replace every non-letter in each sentence by space.
2. Then split on spaces.

- **Question:** How do we specify all non-letter characters?

Function replaceNonLetters

```
# Replaces all non-letters in a given string s by space
def replaceNonLetters(s):
    # Make a list of all non-letters. Note the use of the list comprehension here
    nonLetters = [x for x in s if not x.isalpha()]

    # Replaces each nonletter character in s by space
    for char in nonLetters:
        s = s.replace(char, " ")

    return s
```

Function replaceNonLetters: Alternate version

```
def nonLetterToSpace(ch):
    if ch.isalpha():
        return ch
    else:
        return " "

def replaceNonLetters(s):
    return "".join([nonLetterToSpace(x) for x in s])
```

- Note the use of a **join** in conjunction with a list comprehension.

Function parseWords

```
# Takes a list of sentences and parses each sentence in this list into a list of words.
# So the result is a list of lists, e.g., [["This", "is", "ok"], ["This", "is", "not"]].
# We use the same definition of a word as before. It is a contiguous sequence of
# letters.

def parseWords(sentenceList):

    # Once non-letters have been replaced by spaces then a simple split() using
    # blank as the delimiter will help us get all the words. Note that this
    # constructs a nested list of words for each sentence.

    return [replaceNonLetters(x).split() for x in sentenceList]
```

Part 1 of the main program

```
# main program
f = open("illiad.txt", "r")
bigString = f.read()
sentenceList = parseSentences(bigString)
nestedWordList = parseWords(sentenceList)
```

- This produces a list of word lists from the file, where each word list corresponds to words in a sentence.

- **Example:** [["This", "is", "ok"], ["This", "is", "not"]]

Part 2 of the main program

```
# This block of code creates a new nested list of word lists with the
# the first word in each sentence deleted. Then this nested list of word
# lists is flattened into a list of words. Finally, from this list we pick words
# that start with an upper case letter and have length at least 4 and create
# a new list.

# We then use computeFrequencies (remember, from HW4) to produce a list
# of unique words and their frequencies.

nestedWordList = [x[1:] for x in nestedWordList]
wordList = [y for x in nestedWordList for y in x]
characterNames = [x for x in wordList if x[0].isupper() and len(x) > 3]

[masterList, frequencies] = computeFrequencies(characterNames)
```

Part 3 of the main program

```
# zips the frequencies and words together and sorts the zipped list in descending
# order of frequencies.
combinedList = [[frequencies[i], masterList[i]] for i in range(len(masterList))]
combinedList.sort(reverse = True)

# Prints the 30 most frequent character names
print(combinedList[:30])
```

- Note that `combinedList` is a list of ordered pairs, each ordered pair being a list with a frequency followed by a word. This is because we are sorting (in non-increasing order) of frequency.