String Operations

MARCH 27TH 2013

Python has lots of string operations…

- You can find a bunch of these in Section 5.6.1. “String Methods” of the Python documentation (v. 2.7.2).

- These are in addition to the operations we studied that are common to lists *and* strings.
  - indexing, slicing
  - membership testing (`in` and `not in`) and concatenation (`+`).
  - `index`, `count`

String operations

- Here is a categorization (of some of these methods) that might help you navigate the long list of available string operations:
  - **Boolean methods**: isalpha, isalnum, isdigit, islower, isupper, isspace , startswith.
  - **Reformatting methods**: lower, upper, swapcase, capitalize, center, strip, lstrip, rstrip, ljust, rjust.
  - **Split methods**: split, lsplit, rsplit, splitlines.
  - **Join methods**: join.
  - **Replace methods**: replace

Examples: boolean methods

```
>>> "hello".isalpha()
True
>>> "hello".isalnum()
True
>>> "1234".isdigit()
True
>>> "39.78".isalnum()
False
>>> "hello?".islower()
True
>>> "Hello??".islower()
False
>>> "hello?".startswith("he")
True
>>> "hello?".startswith("He")
False
```

Examples: Reformatting methods

```
>>> "Hello, how are you?".lower()
'hello, how are you?'
>>> "Hello, how are you?".swapcase()
'hELLO, HOW ARE YOU?'
>>> "jack".capitalize()
'Jack'
>>> "  this string has spaces..   ".strip()
'this string has spaces..'
>>> "  this string has spaces..   ".lstrip()
'this string has spaces..   '
>>> "  this string has spaces..   ".rstrip()
'  this string has spaces..'
>>> "test".center(20)
'        test        '
>>> "hello??".rjust(20)
'             hello??'
```

Split and Join

```
>>> "hello, how are you?".split()
['hello,', 'how', 'are', 'you?']
>>> "Other, characters, can, be, used,,to,split?".split(",")
['Other', ' characters', ' can', ' be', ' used', '', 'to',
'split?']
>>> '''This string
... spans a
... few lines'''.splitlines()
['This string', 'spans a ', 'few lines']
>>> " ".join(["hello", "are","you","ok?"])
'hello are you ok?'
>>> "??".join(["hello", "are","you","ok?"])
'hello??are??you??ok?'
```

Replace

```
>>> "hello how are you?".replace(" ", "!")
'hello!how!are!you?'
>>> "hello, how are you?".replace("h","")
'ello, ow are you?'
```

Problem

Let us use our recently acquired knowledge about list comprehensions and string methods to solve the following problem.

Given a novel (e.g., “War and Peace”) find the principal characters in the novel.

General Approach

- Each character’s name appears in the text as a proper noun and hence starting with an upper case letter.

- Words that start *sentences* always start with upper case letters, so we should ignore these.

- We can define a sentence as a sequence of characters delimited by the punctuation marks “.”, “!”, and “?”. (There might be more sophisticated ways of defining sentences.)

General Approach (continued)

- So we start by partitioning the text into sentences.
- We then partition each sentence into words.
- We then count those words that start with an upper case letter and do not start a sentence.
- We maintain frequencies (as in Homework 3) and report the most words gathered in this manner.

Output

- When run on “War and Peace” the output was:

```
[(1478, 'Pierre'), (1208, 'Prince'), (1124, 'Andrew'),
(886, 'Natasha'), (878, 'French'), (703, 'Moscow'),
(645, 'Mary'), (625, 'Emperor'), (591, 'Rostov'),
(495, 'Nicholas'), (488, 'Napoleon'), (453, 'Russian'),
(445, 'Princess'), (429, 'Kutuzov'), (343, 'Denisov'),
(330, 'Sonya'), (257, 'Dolokhov'), (243, 'Petersburg'),
(240, 'Count'), (238, 'Vasili')]
```

Parsing the text into sentences

```
# Takes a string as parameter and "splits" it into "sentences."
# We assume that ".", "!", and "?" are sentence delimiters
def parseSentences(bigString):
    # First split using ".". This creates a list of sentences, which need to
    # be further split using "!" and "?"
    sentenceList = bigString.split(".")

    # For each delimiter that is either "?" or "!", split according to
    # that delimiter
    for delimiter in ["?", "!"]:
        sentenceList = [x.split(delimiter) for x in sentenceList]

        # This creates a nested list, that needs to be flattened. We use a list
        # comprehension to flatten the list.
        sentenceList = [y for x in sentenceList for y in x]

    return sentenceList
```

Parsing a list of sentences into lists of words

```
# 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):
    # Make a list of all non-letters. Note the use of the list comprehension here
    nonLetters = [chr(x) for x in range(0, ord("A")) + range(ord("Z")+1, ord("a")) + range(ord("z")
+1, 127)]

    # Replaces each non-letter character in each sentence by a blank
    for i in range(len(sentenceList)):
        for char in nonLetters:
            sentenceList[i] = sentenceList[i].replace(char, " ")

    # Once non-letters have beeb replaced by blanks 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.
    nestedWordList = [x.split() for x in sentenceList]
    return nestedWordList
```

The main program

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

# This block of code walks through the list of words, ignores
# the first word in each sentence and of the remaining words, picks
# ones that start with an upper case and have length at least 4.
characterNames = []
for sentenceWords in nestedWordList:
    restOfWords = sentenceWords[1:]
    for w in restOfWords:
        if w[0].isupper() and len(w) > 3:
            characterNames.append(w)

[masterList, frequencies] = computeFrequencies(characterNames)

# zips the frequencies and words together and sorts the zipped list in descending
# order of frequencies.
combinedList = zip(frequencies, masterList)
combinedList.sort(reverse = True)

# Prints the 20 most frequent character names
print combinedList[:20]
f.close()
```