Design Exercise

UW CSE 160

Winter 2016

Exercise

Given a problem description, design a module to solve the problem

1) Specify a set of functions
   - For each function, provide
     - the name of the function
     - a doc string for the function
2) Sketch an implementation of each function
   - In English, describe what the implementation needs to do
   - This will typically be no more than about 4-5 lines per function

Example of high-level "pseudocode"

```
def read_scores(filename)
  """Read scores from filename and return a dictionary mapping words to scores"""
  open the file
  For each line in the file,
     insert the word and its score into a dictionary called scores
  return the scores dictionary

def compute_total_sentiment(searchterm):
  """Return the total sentiment for all words in all tweets in the first page of results
returned for the search term"""
  Construct the twitter search url for searchterm
  Fetch the twitter search results using the url
  For each tweet in the response,
     extract the text
    add up the scores for each word in the text
    add the score to the total
  return the total
```

Exercise 1: Text analysis

Design a module for basic text analysis with the following capabilities:

- Compute the total number of words in a file
- Find the 10 most frequent words in a file.
- Find the number of times a given word appears in the file.

Also show how to use the interface by computing the top 10 most frequent words in the file **`testfile.txt`**

Text Analysis, Version 1

```
def word_count(filename, word):
    """Given a filename and a word, return the count
of the given word in the given file."""

def top10(filename):
    """Given a filename, return a list of the top 10
most frequent words in the given file, from most
frequent to least frequent."""

def total_words(filename):
    """Given a filename, return the total number of
words in the file."""

# program to compute top 10:
result = top10("somedocument.txt")
```

- Pros:

- Cons:

Text Analysis, Version 2

```
def read_words(filename):
    """Given a filename, return a list of words in the
file."""
def word_count(wordlist, word):
    """Given a list of words and a word, returns a pair
(count, allcounts_dict). count is the number of
occurrences of the given word in the list, allcounts_dict
is a dictionary mapping words to counts."""
def top10(wordcounts_dict):
    """Given a dictionary mapping words to counts, return
a list of the top 10 most frequent words in the
dictionary, from most to least frequent."""
def total_words(wordlist):
    """Return total number of words in the given list."""

# program to compute top 10:
word_list = read_words("somedocument.txt")
(count, word_dict) = word_count(word_list, "anyword")
result = top10(word_dict)
```

- Pros:

- Cons:

Text Analysis, Version 3

```
def read_words(filename):
    """Given a filename, return a dictionary mapping
each word in filename to its frequency in the file"""

def word_count(word_counts_dict, word):
    """Given a dictionary mapping word to counts, return
the count of the given word in the dictionary."""

def top10(word_counts_dict):
    """Given a dictionary mapping word to counts, return
a list of the top 10 most frequent words in the
dictionary, from most to least frequent."""

def total_words(word_counts_dict):
 """Given a dictionary mapping word to counts, return
the total number of words used to create the
dictionary"""

# program to compute top 10:
word_dict = read_words("somedocument.txt")
result = top10(word_dict)
```

- Pros:
- Cons:

Analysis

- Consider the 3 designs
- For each design, state positives and negatives
- Which one do you think is best, and why?

Changes to text analysis problem

- Ignore *stopwords* (common words such as "the")
  - A list of stopwords is provided in a file, one per line.

- Show the top *k* words rather than the top 10.

Design criteria

- Ease of use vs. ease of implementation
  - Module may be written once but re-used many times
- Generality
  - Can it be used in a new situation?
  - Decomposability: Can parts of it be reused?
  - Testability: Can parts of it be tested?
- Documentability
  - Can you write a coherent description?
- Extensibility: Can it be easily changed?

From Exercise 1:

```
def read_words(filename):
 """Given a filename, return a dictionary mapping each
word in filename to its frequency in the file"""
    wordfile = open(filename)
    worddata = wordfile.read()
    word_list = worddata.split()
    wordfile.close()
    wordcounts = {}
    for word in word_list:
        if wordcounts.has_key(word):
            wordcounts[word] = wordcounts[word] + 1
        else:
            wordcounts[word] = 1
    return wordcounts
```

This “default” pattern is so common, there is a special method for it.

setdefault

```
def read_words(filename):
 """Given a filename, return a dictionary mapping each
word in filename to its frequency in the file"""
    wordfile = open(filename)
    worddata = wordfile.read()
    word_list = worddata.split()
    wordfile.close()
    wordcounts = {}
    for word in word_list:
        count = wordcounts.setdefault(word, 0)
        wordcounts[word] = count + 1
    return wordcounts
```

This “default” pattern is so common, there is a special method for it.

setdefault

```
for word in word_list:
    if wordcounts.has_key(word):
        wordcounts[word] = wordcounts[word] + 1
    else:
        wordcounts[word] = 1
```

VS:

```
for word in word_list:
    count = wordcounts.setdefault(word, 0)
    wordcounts[word] = count + 1
```

`setdefault(`*key*[, *default*]**`)`**

- If *key* is in the dictionary, return its value.
- If *key* is NOT present, insert *key* with a value of *default*, and return *default*.
- If *default* is not specified, the value **`None`** is used.