Vocabulary Size of Moby Dick

March 5, 2015
The Big Picture

Overall Goal
Build a Concordance of a text
  • Locations of words
  • Frequency of words

Today: Summary Statistics
  • Get the vocabulary size of Moby Dick (Attempt 1)
    • Write test cases to make sure our program works
  • Think of a faster way to compute the vocabulary size
Last Class Activity & Homework

• Let’s review common office hours questions

• Understanding how ‘for loops’ work
  – Using a variable to accumulate some value
    • (e.g., a count, a sum, a list) over the course of running the loop

• Understanding how ‘if statements’ work
  • Do I need the ‘else’ part?
From Last Class

• Finding average word length, longest word
Compute the Average Word Length of Moby Dick

def avgWordLengthInMobyDick():
    '''Gets the average word length in MobyDick.txt'''
    myList = readMobyDick()
    s = 0  # tally of lengths of all words encountered so far
    for word in myList:
        s = s + len(word)
    avg = s/float(len(myList))
    return avg
def avgWordLengthInMobyDick():
    '''Gets the average word length in MobyDick.txt'''
    myList = readMobyDick()
    s = 0  # tally of lengths of all words encountered so far
    for word in myList:
        s = s + len(word)
    avg = s/float(len(myList))
    return avg
def avgWordLengthInMobyDick():
    '''Gets the average word length in MobyDick.txt'''
    myList = readMobyDick()
    s = 0  # tally of lengths of all words encountered so far
    for word in myList:
        s = s + len(word)
    avg = s/float(len(myList))
    return avg
How it works

[“cat”, “puppy”, “dog”, “kitty”]

s: 3
word: puppy

```python
def avgWordLengthInMobyDick():
    '''Gets the average word length in MobyDick.txt'''
    myList = readMobyDick()
    s = 0  # tally of lengths of all words encountered so far
    for word in myList:
        s = s + len(word)
    avg = s/float(len(myList))
    return avg
```
How it works

["cat", "puppy", "dog", "kitty"]

s: 8

word: puppy

def avgWordLengthInMobyDick():
    """Gets the average word length in MobyDick.txt"""
    myList = readMobyDick()
    s = 0  # tally of lengths of all words encountered so far
    for word in myList:
        s = s + len(word)
    avg = s/float(len(myList))
    return avg
How it works

[“cat”, “puppy”, “dog”, “kitty”]

```python
def avgWordLengthInMobyDick():
    '''Gets the average word length in MobyDick.txt'''
    myList = readMobyDick()
    s = 0  # tally of lengths of all words encountered so far
    for word in myList:
        s = s + len(word)
    avg = s/float(len(myList))
    return avg
```

word: dog

s: 8
How it works

[“cat”, “puppy”, “dog”, “kitty”]

s: 11

word: dog

def avgWordLengthInMobyDick():
    ''' Gets the average word length in MobyDick.txt '''
    myList = readMobyDick()
    s = 0  # tally of lengths of all words encountered so far
    for word in myList:
        s = s + len(word)
    avg = s/float(len(myList))
    return avg
How it works

```
["cat", "puppy", "dog", "kitty"]
```

`s:` 11

`word:` kitty

def avgWordLengthInMobyDick():
    '''Gets the average word length in MobyDick.txt'''
    myList = readMobyDick()
    s = 0  # tally of lengths of all words encountered so far
    for word in myList:
        s = s + len(word)
    avg = s/float(len(myList))
    return avg
```
How it works

[“cat”, “puppy”, “dog”, “kitty”]

s: 16

word: kitty

def avgWordLengthInMobyDick():
    '''Gets the average word length in MobyDick.txt'''
    myList = readMobyDick()
    s = 0 # tally of lengths of all words encountered so far
    for word in myList:
        s = s + len(word)
    avg = s/float(len(myList))
    return avg
def avgWordLengthInMobyDick():
    '''Gets the average word length in MobyDick.txt'''
    myList = readMobyDick()
    s = 0  # tally of lengths of all words encountered so far
    for word in myList:
        s = s + len(word)
    avg = s/float(len(myList))
    return avg
Now the longest word...
Get the Longest Word in Moby Dick

def getLongestWordInMobyDick():
    '''Returns the longest word in MobyDick.txt'''
    return longestword
Get the Longest Word in *Moby Dick*

```python
def getLongestWordInMobyDick():
    '''Returns the longest word in MobyDick.txt'''
    myList = readMobyDick()
    longestWord = ""
    for word in myList:
        if len(word) > len(longestWord):
            longestWord = word
    return longestWord
```
Get the Longest Word in Moby Dick

```python
def getLongestWordInMobyDick():
    '''Returns the longest word in MobyDick.txt'''
    myList = readMobyDick()
    longestWord = ""
    for word in myList:
        if len(word) > len(longestWord):
            longestWord = word
    return longestWord
```

Is our program correct?
How it works

[“cat”, “elephant”, “zebra”, “flying squirrel”]

```python
def getLongestWordInMobyDick():
    '''Returns the longest word in MobyDick.txt'''
    myList = readMobyDick()
    longestword = ""
    for word in myList:
        if len(word) > len(longestword):
            longestword = word
    return longestword
```
def getLongestWordInMobyDick():
    '''Returns the longest word in MobyDick.txt'''
    myList = readMobyDick()
    longestword = ""
    for word in myList:
        if len(word) > len(longestword):
            longestword = word
    return longestword
def getLongestWordInMobyDick():
    '''Returns the longest word in MobyDick.txt'''
    myList = readMobyDick()
    longestword = ""
    for word in myList:
        if len(word) > len(longestword):
            longestword = word
    return longestword

longestWord: cat

word: elephant
def getLongestWordInMobyDick():
    '''Returns the longest word in MobyDick.txt'''
    myList = readMobyDick()
    longestword = ""
    for word in myList:
        if len(word) > len(longestword):
            longestword = word
    return longestword
How it works

[“cat”, “elephant”, “zebra”, “flying squirrel”]

```python
def getLongestWordInMobyDick():
    '''Returns the longest word in MobyDick.txt'''
    myList = readMobyDick()
    longestword = ""
    for word in myList:
        if len(word) > len(longestword):
            longestword = word
    return longestword
```
How it works

[“cat”, “elephant”, “zebra”, “flying squirrel”]

longestWord: elephant

word: flying squirrel

def getLongestWordInMobyDick():
    '''Returns the longest word in MobyDick.txt'''
    myList = readMobyDick()
    longestword = ""
    for word in myList:
        if len(word) > len(longestword):
            longestword = word
    return longestword
def getLongestWordInMobyDick():
    '''Returns the longest word in MobyDick.txt'''
    myList = readMobyDick()
    longestword = ""
    for word in myList:
        if len(word) > len(longestword):
            longestword = word
    return longestword
How it works

[“cat”, “elephant”, “zebra”, “flying squirrel”]

```python
def getLongestWordInMobyDick():
    '''Returns the longest word in MobyDick.txt'''
    myList = readMobyDick()
    longestword = ""
    for word in myList:
        if len(word) > len(longestword):
            longestword = word
    return longestword
```

return this

longestWord: flying squirrel

word: flying squirrel
Why use functions?

• Break up tasks into smaller tasks
  – Test smaller tasks; then assemble!
• Functions allow generalization!
Compute the Average Word Length of a word list

def avgWordLength (wordList):
    '''Average word length in a nonempty list of words'''
    s = 0  # tally of lengths of all words encountered so far
    for word in wordList:
        s = s + len(word)
    avg = s/float(len(wordList))  # assumes wordList nonempty!
    return avg
Designing functions

• What constitutes a “smaller task”?  
• Bad choice: “find average word length in first third of Moby Dick” 
• Good choice: “read in list of all words of Moby Dick”; “compute average word length in list” 
• For now...we’ll guide you on this.
ACT2-4

• Do Task 1 – Practice spotting errors in functions
## Debugging Programs

<table>
<thead>
<tr>
<th>Function</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>addTwo ((x, y))</td>
<td>addTwo ((2, 0))</td>
</tr>
<tr>
<td>subtractTwo ((x, y))</td>
<td>subtractTwo ((2, 0))</td>
</tr>
<tr>
<td>multiplyTwo ((x, y))</td>
<td>(z = \text{multiply}(2, 0))</td>
</tr>
<tr>
<td>divideTwo ((x, y))</td>
<td>divideTwo ((2, 0))</td>
</tr>
<tr>
<td>addList (myList)</td>
<td>myList ([2, 0])</td>
</tr>
</tbody>
</table>
Debugging Programs

<table>
<thead>
<tr>
<th>Function</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>addTwo(x, y)</td>
<td>addTwo(2, 0) – output is wrong</td>
</tr>
<tr>
<td>subtractTwo(x, y)</td>
<td>subtractTwo(2, 0) – any input causes an error</td>
</tr>
<tr>
<td>multiplyTwo(x, y)</td>
<td>z = multiply(2, 0) – What is z after this assignment?</td>
</tr>
<tr>
<td>divideTwo(x, y)</td>
<td>divideTwo(2, 0) – What happens when I run this? Use an “if” to catch the error and print a message to the screen.</td>
</tr>
<tr>
<td>addList(myList)</td>
<td>myList([2, 0]) – any input causes an error</td>
</tr>
</tbody>
</table>
## Boolean Expressions on Numbers

<table>
<thead>
<tr>
<th>Expression</th>
<th>Evaluation</th>
<th>Final</th>
</tr>
</thead>
<tbody>
<tr>
<td>((100 &gt; 101) \text{ and } (-1 \neq -1))</td>
<td></td>
<td></td>
</tr>
<tr>
<td>((1 \leq 2) \text{ or } ((1 == 1) \text{ and } (1 \neq 2)))</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(x = 1) \text{ not} (x &gt; 2) \text{ or } (x \leq x+1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(y = 100) \text{ not} \text{not} (y == 100)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Boolean Expressions on Numbers

<table>
<thead>
<tr>
<th>Expression</th>
<th>Evaluation</th>
<th>Final</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>(100 &gt; 101) and (-1 != -1)</code></td>
<td>False and False</td>
<td>False</td>
</tr>
<tr>
<td><code>(1 &lt;= 2) or ((1 == 1) and (1 != 2)</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>x = 1</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>not(x &gt; 2) or (x &lt;= x+1)</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>y = 100</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>not(not(y == 100))</code></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Boolean Expressions on Numbers

<table>
<thead>
<tr>
<th>Expression</th>
<th>Evaluation</th>
<th>Final</th>
</tr>
</thead>
<tbody>
<tr>
<td>(100 &gt; 101) and (-1 != -1)</td>
<td>False and False</td>
<td>False</td>
</tr>
<tr>
<td>(1 &lt;= 2) or ((1 == 1) and (1 != 2)</td>
<td>True or (True and True)</td>
<td>True</td>
</tr>
<tr>
<td>x = 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>not(x &gt; 2) or (x &lt;= x+1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>y = 100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>not(not(y == 100))</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Boolean Expressions on Numbers

<table>
<thead>
<tr>
<th>Expression</th>
<th>Evaluation</th>
<th>Final</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>(100 &gt; 101) and (-1 != -1)</code></td>
<td><code>False and False</code></td>
<td><code>False</code></td>
</tr>
<tr>
<td><code>(1 &lt;= 2) or ((1 == 1) and (1 != 2)</code></td>
<td><code>True or (True and True)</code></td>
<td><code>True</code></td>
</tr>
<tr>
<td><code>x = 1</code> <code>not(x &gt; 2) or (x &lt;= x+1)</code></td>
<td><code>True or True</code></td>
<td><code>True</code></td>
</tr>
<tr>
<td><code>y = 100</code> <code>not(not(y == 100))</code></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Boolean Expressions on Numbers

<table>
<thead>
<tr>
<th>Expression</th>
<th>Evaluation</th>
<th>Final</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>(100 &gt; 101) and (-1 != -1)</code></td>
<td><code>False and False</code></td>
<td><code>False</code></td>
</tr>
<tr>
<td><code>(1 &lt;= 2) or ((1 == 1) and (1 != 2)</code></td>
<td><code>True or (True and True)</code></td>
<td><code>True</code></td>
</tr>
<tr>
<td><code>x = 1</code> not(x &gt; 2) or (x &lt;= x+1)`</td>
<td><code>True or True</code></td>
<td><code>True</code></td>
</tr>
<tr>
<td><code>y = 100</code> not(not(y == 100))</td>
<td><code>not(False)</code></td>
<td><code>True</code></td>
</tr>
</tbody>
</table>
## Boolean Expressions on Strings

<table>
<thead>
<tr>
<th>Operator</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equality</td>
<td>'a' == 'b'</td>
<td>False</td>
</tr>
<tr>
<td>Inequality</td>
<td>'a' != 'b'</td>
<td>True</td>
</tr>
</tbody>
</table>
## Boolean Expressions on Strings

### Boolean Operators on Strings

<table>
<thead>
<tr>
<th>Operator</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equality</td>
<td>'a' == 'b'</td>
<td>False</td>
</tr>
<tr>
<td>Inequality</td>
<td>'a' != 'b'</td>
<td>True</td>
</tr>
</tbody>
</table>

```python
gt 3
>>> 'apple' == 'apple'
True
>>> 'apple' == 'Apple'
False
>>> 'apple' == 'apple!'
False
```
The Big Picture

Overall Goal
Build a Concordance of a text
  • Locations of words
  • Frequency of words

Today: Summary Statistics
• Get the vocabulary size of Moby Dick (Attempt 1)
  • Write test cases to make sure our program works
• Think of a faster way to compute the vocabulary size

Save ACT2-4.py and MobyDick.txt to the same directory
def vocabSize():
    myList = readMobyDickShort()
    uniqueList = noReplicates(myList)
    return len(uniqueList)
Writing a vocabSize Function

def vocabSize():
    myList = readMobyDickShort()
    uniqueList = noReplicates(myList)
    return len(uniqueList)

def noReplicates(wordList):
    '''takes a list as argument, returns a list free of replicate items. slow implementation.'''

def isElementOf(myElement, myList):
    '''takes a string and a list and returns True if the string is in the list and False otherwise.'''
Writing a `vocabSize` Function

```python
def vocabSize():
    myList = readMobyDickShort()
    uniqueList = noReplicates(myList)
    return len(uniqueList)
```

```python
def noReplicates(wordList):
    """takes a list as argument, returns a list free of replicate items. slow implementation."
```

```python
def testNoReplicates():
```

```python
def isElementOf(myElement, myList):
    """takes a string and a list and returns True if the string is in the list and False otherwise."
```

```python
def testIsElementOf():
```

Writing test cases is important to make sure your program works!
The Big Picture

Overall Goal
Build a Concordance of a text
- Locations of words
- Frequency of words

Today: Summary Statistics
- Get the vocabulary size of Moby Dick (Attempt 1)
  - Write test cases to make sure our program works
- Think of a faster way to compute the vocabulary size
What does slow implementation mean?

• Replace `readMobyDickShort()` with `readMobyDickAll()`

• Now, run `vocabSize()`
  – Hint: Ctrl-C (or Command-C) will abort the call.
What does slow implementation mean?

• Replace `readMobyDickShort()` with `readMobyDickAll()`

• Now, run `vocabSize()`
  – Hint: Ctrl-C (or Command-C) will abort the call.

• *Faster way to write* `noReplicates()`
  – What if we can sort the list?
    ```python
    ['a','a','a','at','and','and',...,'zebra']
    ```
## Sorting Lists

<table>
<thead>
<tr>
<th>Name</th>
<th>Inputs</th>
<th>Outputs</th>
<th>CHANGES</th>
</tr>
</thead>
<tbody>
<tr>
<td>sort</td>
<td>List</td>
<td>Original List!</td>
<td></td>
</tr>
</tbody>
</table>
## Sorting Lists

### Preloaded Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Inputs</th>
<th>Outputs</th>
<th>CHANGES</th>
</tr>
</thead>
<tbody>
<tr>
<td>sort</td>
<td>List</td>
<td>Original List!</td>
<td></td>
</tr>
</tbody>
</table>

```python
>>> myList = [0,4,1,5,-1,6]
>>> myList.sort()
>>> myList
[-1, 0, 1, 4, 5, 6]
```
# Sorting Lists

<table>
<thead>
<tr>
<th>Preloaded Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
</tr>
<tr>
<td>sort</td>
</tr>
</tbody>
</table>

```python
>>> myList = [0, 4, 1, 5, -1, 6]
>>> myList.sort()
>>> myList
[-1, 0, 1, 4, 5, 6]

>>> myList = ['b', 'd', 'c', 'a', 'z', 'i']
>>> myList.sort()
>>> myList
['a', 'b', 'c', 'd', 'i', 'z']
```
The Big Picture

Overall Goal
Build a Concordance of a text
  • Locations of words
  • Frequency of words

Today: Summary Statistics
• Get the vocabulary size of Moby Dick (Attempt 1)
  • Write test cases to make sure our program works
• Think of a faster way to compute the vocabulary size