Vocabulary Size of Moby Dick

Oct 15, 2015
The Big Picture

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

From Last Lecture: Summary Statistics
• Nitpicky Python details
• A new kind of statement
• Count the number of words in Moby Dick
• Compute the average word length of Moby Dick
• Find the longest word in Moby Dick
• Get the vocabulary size of Moby Dick
How does Python know what's a variable?

- A **literal** is a piece of data that we give directly to Python
  - 'hello' is a string (str) literal
  - So are "hey there" and 'what\'s up'
  - 5 is an integer (int) literal
  - 32.8 is a floating-point (float) literal
"How does Python know what's a variable?"

• Variable names are made up of:
  – Letters (uppercase and lowercase)
  – Numbers (but only after the first letter)
  – Underscores

• Names for functions and types follow the same rules

• Anything else must be a literal or operator!
Using String Literals

def getFile(fnRelative):
    '''Opens the appropriate file in my folder'''
    fnAbsolute = "/Users/alexandra/" + fnRelative
    return open(fnAbsolute, "r")

myFile = getFile("MobyDick.txt")
Using Functions

```python
def addOneBAD(t):
    t = t + 1
    return t

def addOneGOOD(t):
    x = t + 1
    return x
```

*Do not change argument values inside your functions;*

*Use new variables instead*
The Big Picture

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

From Last Lecture: Summary Statistics
• Administrative stuff
• Nitpicky Python details
• A new kind of statement
• Count the number of words in Moby Dick
• Compute the average word length of Moby Dick
• Find the longest word in Moby Dick
• Get the vocabulary size of Moby Dick
Review: Basic Types

- Integers
  - 3
  - -100
  - 1234

- Floats
  - 12.7
  - -99.99
  - 1234.0

- Strings
  - '12'
  - 'hi'
  - 'Moby'

- Booleans
  - True
  - False

New literals representing... truth and falseness
New Type: Booleans

• Either True or False
  – Note the capitalization

```python
>>> x = True
>>> x
True
>>> y = False
>>> y
False
```
New Type: Booleans

• Either **True** or **False**
  – Note the capitalization

• New Operators

Remember

<table>
<thead>
<tr>
<th>Numerical Operators</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Operator</strong></td>
</tr>
<tr>
<td>Sum</td>
</tr>
<tr>
<td>Difference</td>
</tr>
</tbody>
</table>
New Type: Booleans

- Either **True** or **False**
  - Note the capitalization

- New Operators

Remember

<table>
<thead>
<tr>
<th>Numerical Operators</th>
<th>Operator</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sum</td>
<td>1 + 2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Difference</td>
<td>1 - 2</td>
<td>-1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Boolean Operators</th>
<th>Operator</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equality</td>
<td>1 == 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Inequality</td>
<td>1 != 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Less Than</td>
<td>1 &lt; 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Less Than or Equal To</td>
<td>1 &lt;= 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Greater Than</td>
<td>1 &gt; 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Greater Than or Equal To</td>
<td>1 &gt;= 2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
New Type: Booleans

- Either True or False
  - Note the capitalization
- New Operators

<table>
<thead>
<tr>
<th>Numerical Operators</th>
<th>Operator</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sum</td>
<td>1 + 2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Difference</td>
<td>1 - 2</td>
<td>-1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Boolean Operators</th>
<th>Operator</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equality</td>
<td>1 == 2</td>
<td>False</td>
<td></td>
</tr>
<tr>
<td>Inequality</td>
<td>1 != 2</td>
<td>True</td>
<td></td>
</tr>
<tr>
<td>Less Than</td>
<td>1 &lt; 2</td>
<td>True</td>
<td></td>
</tr>
<tr>
<td>Less Than or Equal To</td>
<td>1 &lt;= 2</td>
<td>True</td>
<td></td>
</tr>
<tr>
<td>Greater Than</td>
<td>1 &gt; 2</td>
<td>False</td>
<td></td>
</tr>
<tr>
<td>Greater Than or Equal To</td>
<td>1 &gt;= 2</td>
<td>False</td>
<td></td>
</tr>
</tbody>
</table>
New Type: Booleans

- Either **True** or **False**
  - Note the capitalization
- New Operators
- These are expressions
- Assignments have only one equals sign.

<table>
<thead>
<tr>
<th>Boolean Operators</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equality</td>
<td>1 == 2</td>
<td>False</td>
</tr>
<tr>
<td>Inequality</td>
<td>1 != 2</td>
<td>True</td>
</tr>
<tr>
<td>Less Than</td>
<td>1 &lt; 2</td>
<td>True</td>
</tr>
<tr>
<td>Less Than or Equal To</td>
<td>1 &lt;= 2</td>
<td>True</td>
</tr>
<tr>
<td>Greater Than</td>
<td>1 &gt; 2</td>
<td>False</td>
</tr>
<tr>
<td>Greater Than or Equal To</td>
<td>1 &gt;= 2</td>
<td>False</td>
</tr>
</tbody>
</table>
Boolean Types

Last Boolean Operators: **and**, **or** and **not**

<table>
<thead>
<tr>
<th>Boolean Operators</th>
<th>Examples</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>and</strong></td>
<td>(4&lt;5) and (6&lt;3)</td>
<td></td>
</tr>
<tr>
<td><strong>or</strong></td>
<td>(4&lt;5) or (6&lt;3)</td>
<td></td>
</tr>
<tr>
<td><strong>not</strong></td>
<td>not (4&lt;5)</td>
<td></td>
</tr>
</tbody>
</table>
Boolean Types

Last Boolean Operators: and, or and not

<table>
<thead>
<tr>
<th>Boolean Operators</th>
<th>Operator</th>
<th>Examples</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>and</td>
<td>(4&lt;5) and (6&lt;3)</td>
<td>True and False</td>
</tr>
<tr>
<td></td>
<td>or</td>
<td>(4&lt;5) or (6&lt;3)</td>
<td>True or False</td>
</tr>
<tr>
<td></td>
<td>not</td>
<td>not (4&lt;5)</td>
<td>not (True)</td>
</tr>
</tbody>
</table>
Boolean Types

Last Boolean Operators: and, or and not

<table>
<thead>
<tr>
<th>Boolean Operators</th>
<th>Operator</th>
<th>Examples</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>and</td>
<td>(4&lt;5) and (6&lt;3)</td>
<td>True and False</td>
</tr>
<tr>
<td></td>
<td>or</td>
<td>(4&lt;5) or (6&lt;3)</td>
<td>True or False</td>
</tr>
<tr>
<td></td>
<td>not</td>
<td>not (4&lt;5)</td>
<td>not (True)</td>
</tr>
</tbody>
</table>
# Boolean Types

**Last Boolean Operators:** and, or and not

## Boolean Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Examples</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>and</td>
<td>(4&lt;5) and (6&lt;3)</td>
<td>True and False</td>
</tr>
<tr>
<td></td>
<td></td>
<td>False</td>
</tr>
<tr>
<td>or</td>
<td>(4&lt;5) or (6&lt;3)</td>
<td>True or False</td>
</tr>
<tr>
<td></td>
<td></td>
<td>True</td>
</tr>
<tr>
<td>not</td>
<td>not(4&lt;5)</td>
<td>not(True)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>False</td>
</tr>
</tbody>
</table>

## More Examples

<table>
<thead>
<tr>
<th>Examples</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>(4&lt;5) and ((6&lt;3) or (5==5))</td>
<td></td>
</tr>
<tr>
<td>(5==4) or (not(6&lt;3))</td>
<td></td>
</tr>
</tbody>
</table>
Boolean Types

Last Boolean Operators: and, or and not

<table>
<thead>
<tr>
<th>Boolean Operators</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Operator</strong></td>
</tr>
<tr>
<td>and</td>
</tr>
<tr>
<td>or</td>
</tr>
<tr>
<td>not</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>More Examples</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>(4&lt;5) and ((6&lt;3) or (5==5))</td>
<td>True and (False or True)</td>
</tr>
<tr>
<td>(5==4) or (not(6&lt;3))</td>
<td></td>
</tr>
</tbody>
</table>
### Boolean Types

#### Last Boolean Operators: `and`, `or` and `not`

<table>
<thead>
<tr>
<th>Boolean Operators</th>
<th>Examples</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Operator</strong></td>
<td><strong>Examples</strong></td>
<td><strong>Result</strong></td>
</tr>
<tr>
<td><code>and</code></td>
<td><code>(4&lt;5)</code> and <code>(6&lt;3)</code></td>
<td><code>True</code> and <code>False</code></td>
</tr>
<tr>
<td><code>or</code></td>
<td><code>(4&lt;5)</code> or <code>(6&lt;3)</code></td>
<td><code>True</code> or <code>False</code></td>
</tr>
<tr>
<td><code>not</code></td>
<td><code>not(4&lt;5)</code></td>
<td><code>not(True)</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>More Examples</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>(4&lt;5)</code> and <code>((6&lt;3)</code> or <code>(5==5))</code></td>
<td><code>True</code> and <code>(False</code> or <code>True)</code></td>
</tr>
<tr>
<td><code>(5==4)</code> or <code>not(6&lt;3)</code></td>
<td></td>
</tr>
</tbody>
</table>

---

CSCI 0931 - Intro. to Comp. for the Humanities and Social Sciences
# Boolean Types

**Last Boolean Operators:** and, or and not

## Boolean Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Examples</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>and</td>
<td>(4&lt;5) and (6&lt;3)</td>
<td>True and False</td>
</tr>
<tr>
<td></td>
<td></td>
<td>False</td>
</tr>
<tr>
<td>or</td>
<td>(4&lt;5) or (6&lt;3)</td>
<td>True or False</td>
</tr>
<tr>
<td></td>
<td></td>
<td>True</td>
</tr>
<tr>
<td>not</td>
<td>not(4&lt;5)</td>
<td>not(True)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>False</td>
</tr>
</tbody>
</table>

## More Examples

<table>
<thead>
<tr>
<th></th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>(4&lt;5) and ((6&lt;3) or (5==5))</td>
<td>True and (False or True)</td>
</tr>
<tr>
<td>(5==4) or (not(6&lt;3))</td>
<td>False or not(False)</td>
</tr>
</tbody>
</table>
Boolean Types

Last Boolean Operators: and, or and not

<table>
<thead>
<tr>
<th>Boolean Operators</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Operator</strong></td>
</tr>
<tr>
<td>and</td>
</tr>
<tr>
<td>or</td>
</tr>
<tr>
<td>not</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>More Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>(4&lt;5) and ((6&lt;3) or (5==5))</td>
</tr>
<tr>
<td>(5==4) or (not(6&lt;3))</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>
<tr>
<td>Less Than</td>
<td>'a' &lt; 'b'</td>
<td>True</td>
</tr>
<tr>
<td>Less Than or Equal To</td>
<td>'a' &lt;= 'b'</td>
<td>True</td>
</tr>
<tr>
<td>Greater Than</td>
<td>'a' &gt; 'b'</td>
<td>False</td>
</tr>
<tr>
<td>Greater Than or Equal To</td>
<td>'a' &gt;= 'b'</td>
<td>False</td>
</tr>
</tbody>
</table>
Review: Statements

• Expression Statements
  Calculates something

• Assignment Statements
  Stores a value for a variable in memory table

• List-Assignment Statements
  Replaces
  An item or slices of an existing list with new value(s)

• For Statements
  “For each element in myList, do something”

• If Statements
  If A is true, then do something, otherwise do something else
Boolean Statements (If Stmts)

• “If something’s true, do A”

```python
def compare(x, y):
    if x > y:
        print(x, ' is greater than ', y)
```
Boolean Statements (If Stmts)

• “If something’s true, do A, otherwise, do B”

```python
def compare(x, y):
    if x > y:
        print(x, ' is greater than ', y)
    else:
        print(x, ' is less than or equal to ', y)
```
Boolean Statements (If Stmts)

• “If something’s true, do A, otherwise, check something else; if that's true, do B, otherwise, do C”

```python
def compare(x, y):
    '  
    if x > y:
        print(x, ' is greater than ', y)
    else:
        if x < y:
            print(x, ' is less than ', y)
        else:
            print(x, ' is equal to ', y)
```
Boolean Statements (If Stmts) shorthand!

- “If something’s true, do A, otherwise, check something else; if that's true, do B, otherwise, do C”

```python
def compare(x, y):
    """"
    if x > y:
        print(x, ' is greater than ', y)
    elif x < y:
        print(x, ' is less than ', y)
    else:
        print(x, ' is equal to ', y)
```
Review: Other Things

• Lists (a type of **data structure**)

```
[0,1,2]   ['hi','there']   ['hi',0.0]
[1,2,3,4,5,True,False,`true','one']
```
Review: Other Things

• Lists (a type of **data structure**)

```python
[0,1,2]  ['hi','there']  ['hi',0.0]
[1,2,3,4,5,True,False,`true','one']
```

• Files (an **object** that we can open, read, close)

```python
myFile = open(fileName,`r`)```

The Big Picture

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

From Last Lecture: Summary Statistics
• Administrative stuff
• Nitpicky Python details
• A new kind of statement
• Count the number of words in *Moby Dick*
• Compute the average word length of *Moby Dick*
• Find the longest word in *Moby Dick*
## Python Functions

### Preloaded Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>len</code></td>
<td>List</td>
<td>Integer</td>
</tr>
<tr>
<td><code>len</code></td>
<td>String</td>
<td>Integer</td>
</tr>
<tr>
<td><code>len</code></td>
<td>...</td>
<td>Integer</td>
</tr>
</tbody>
</table>

```python
>>> len([3, 47, 91, -6, 18])

>>> uselessList = ['contextless', 'words']
>>> len(uselessList)

>>> creature = 'woodchuck'
>>> len(creature)
```
Python Functions

<table>
<thead>
<tr>
<th>Preloaded Functions</th>
<th>len</th>
<th>List  OR String</th>
<th>Integer</th>
</tr>
</thead>
</table>

# Python Functions

<table>
<thead>
<tr>
<th>Preloaded Functions</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>len</code></td>
<td>List OR String OR ...</td>
</tr>
<tr>
<td><code>float</code></td>
<td>Number (as an Integer, Float, or String)</td>
</tr>
<tr>
<td><code>int</code></td>
<td>Number (as an Integer, Float, or String)</td>
</tr>
<tr>
<td><code>str</code></td>
<td>Integer, Float, String, or List</td>
</tr>
</tbody>
</table>

These functions *cast* a variable of one type to another type.
# Python Functions

<table>
<thead>
<tr>
<th>Preloaded Functions</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>len</code></td>
<td>List OR String OR ...</td>
<td>Integer</td>
</tr>
<tr>
<td><code>float</code></td>
<td>Number (as an Integer, Float, or String)</td>
<td>Float</td>
</tr>
<tr>
<td><code>int</code></td>
<td>Number (as an Integer, Float, or String)</td>
<td>Integer</td>
</tr>
<tr>
<td><code>str</code></td>
<td>Integer, Float, String, or List</td>
<td>String</td>
</tr>
<tr>
<td><code>range</code></td>
<td>Two Integers 1. Start Index (Inclusive) 2. End Index (Exclusive)</td>
<td>List of Integers</td>
</tr>
</tbody>
</table>

These functions cast a variable of one type to another type.
The Big Picture

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

Today: Summary Statistics
• Administrative stuff
• Nitpicky Python details
• A new kind of statement
• Count the number of words in *Moby Dick*
• Compute the average word length of *Moby Dick*
• Find the longest word in *Moby Dick*
ACT2-3

• Do Task 1
ACT2-3

• Do Task 2
ACT2-3

• Do Task 3
The Big Picture

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

Today: Summary Statistics
  • Administrative stuff
  • Nitpicky Python details
  • A new kind of statement
  • Count the number of words in Moby Dick
  • Compute the average word length of Moby Dick
  • Find the longest word in Moby Dick
Get the Longest Word in Moby Dick

def getLongestWordInMobyDick():
    '''Returns the longest word in MobyDick.txt'''
    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
The Big Picture

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

Today: Summary Statistics
• Administrative stuff
• Nitpicky Python details
• A new kind of statement
• Count the number of words in Moby Dick
• Compute the average word length of Moby Dick
• Find the longest word in Moby Dick
Compute the Average Word Length of Moby Dick

```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/ len(myList)
    return avg
```
How it works

[“cat”, “puppy”, “dog”, “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 / len(myList)
    return avg
```
How it works

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

s: 3
word: cat

```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/ 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/ 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/ len(myList)
    return avg
How it works

[“cat”, “puppy”, “dog”, “kitty”]

s: 8

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/ len(myList)
    return avg
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/ 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/ 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 / len(myList)
    return avg
How it works

[“cat”, “puppy”, “dog”, “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/ 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
```
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”]

```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”]

```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
```

longestWord: cat

word: elephant
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
```

longestWord: elephant

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
```

longestWord: elephant
word: flying squirrel
How it works

[“cat”, “elephant”, “zebra”, “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
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
```

```
["cat", "elephant", "zebra", "flying squirrel"]
```

```
return this
```

longestWord: flying squirrel

word: flying squirrel

return this
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

```python
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/ 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><code>addTwo(x, y)</code></td>
<td><code>addTwo(2, 0)</code></td>
</tr>
<tr>
<td><code>subtractTwo(x, y)</code></td>
<td><code>subtractTwo(2, 0)</code></td>
</tr>
<tr>
<td><code>multiplyTwo(x, y)</code></td>
<td><code>z = multiply(2, 0)</code></td>
</tr>
<tr>
<td><code>divideTwo(x, y)</code></td>
<td><code>divideTwo(2, 0)</code></td>
</tr>
<tr>
<td><code>addList(myList)</code></td>
<td><code>myList([2, 0])</code></td>
</tr>
</tbody>
</table>
# Debugging Programs

## Example Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>addTwo(x, y)</code></td>
<td><code>addTwo(2, 0) – output is wrong</code></td>
</tr>
<tr>
<td><code>subtractTwo(x, y)</code></td>
<td><code>subtractTwo(2, 0) – any input causes an error</code></td>
</tr>
<tr>
<td><code>multiplyTwo(x, y)</code></td>
<td><code>z = multiply(2, 0) – What is z after this assignment?</code></td>
</tr>
<tr>
<td><code>divideTwo(x, y)</code></td>
<td><code>divideTwo(2, 0) – What happens when I run this? Use an “if” to catch the error and print a message to the screen.</code></td>
</tr>
<tr>
<td><code>addList(myList)</code></td>
<td><code>myList([2, 0]) – any input causes an error</code></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></td>
<td></td>
</tr>
<tr>
<td>(1 &lt;= 2) or ((1 == 1) and (1 != 2)</td>
<td></td>
<td></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>(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></td>
<td></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>((100 &gt; 101) \text{ and } (-1 \neq -1))</td>
<td>(\text{False and False})</td>
<td>(\text{False})</td>
</tr>
<tr>
<td>((1 \leq 2) \text{ or } ((1 == 1) \text{ and } (1 \neq 2)))</td>
<td>(\text{True or (True and True)})</td>
<td>(\text{True})</td>
</tr>
<tr>
<td>(x = 1) \newline (\text{not}(x &gt; 2) \text{ or } (x \leq x+1))</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(y = 100) \newline (\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>$(100 &gt; 101)$ and $(-1 != -1)$</td>
<td>$\text{False and False}$</td>
<td>$\text{False}$</td>
</tr>
<tr>
<td>$(1 &lt;= 2)$ or $((1 == 1)$ and $(1 != 2)$</td>
<td>$\text{True or (True and True)}$</td>
<td>$\text{True}$</td>
</tr>
<tr>
<td>$x = 1$</td>
<td>$\text{not}(x &gt; 2)$ or $(x &lt;= x+1)$</td>
<td>$\text{True or True}$</td>
</tr>
<tr>
<td>$y = 100$</td>
<td>$\text{not}(\text{not}(y == 100))$</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></td>
<td></td>
<td></td>
</tr>
<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>True or True</td>
<td>True</td>
</tr>
<tr>
<td>not((x &gt; 2)) or ((x &lt;= x+1))</td>
<td>True or True</td>
<td>True</td>
</tr>
<tr>
<td>(y = 100)</td>
<td>not(False)</td>
<td>True</td>
</tr>
<tr>
<td>not(not((y == 100)))</td>
<td></td>
<td></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
>>> '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
Writing a `vocabSize` Function

```python
def vocabSize():
    myList = readMobyDickShort()
    uniqueList = noReplicates(myList)
    return len(uniqueList)
```
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():
    def testIsElementOf(myElement, myList):
        '''takes a string and a list and returns True if the string is in the list and False otherwise.'''
```

```python
def testIsElementOf():
    def testIsElementOf():
        def testIsElementOf():
            def testIsElementOf():
                def testIsElementOf():
                    def testIsElementOf():
                        def testIsElementOf():
                            def testIsElementOf():
                                def testIsElementOf():
                                    def testIsElementOf():
                                        def testIsElementOf():
                                            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?
    ```
    ['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></td>
<td>Original List!</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>Name</th>
<th>Inputs</th>
<th>Outputs</th>
<th>CHANGES</th>
</tr>
</thead>
<tbody>
<tr>
<td>sort</td>
<td>List</td>
<td></td>
<td>Original List!</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