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

Literals vs. Variables

"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

Literals vs. Variables

"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

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

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

t is a parameter, not a "scratchpad"

Another variable (say x) may be your "scratchpad" variable

Do not change argument values inside your functions;

Use new variables instead

The Big Picture

Overall Goal

Build a Concordance of a text

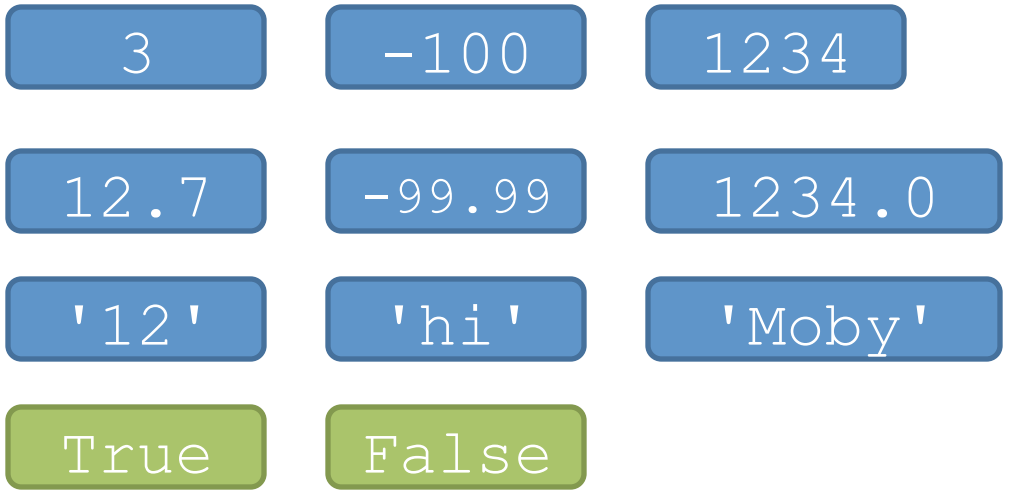
- *Locations* of words
- *Frequency* of words

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
- Floats
- Strings
- Booleans



New literals representing… truth and falseness

New Type: Booleans

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

```
>>> x = True
>>> x
True
>>> y = False
>>> y
False
```

New Type: Booleans

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

Remember

Numerical Operators		
Operator	**Example**	**Result**
Sum	`1 + 2`	`3`
Difference	`1 - 2`	`-1`

New Type: Booleans

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

Remember

Numerical Operators		
Operator	**Example**	**Result**
Sum	`1 + 2`	`3`
Difference	`1 - 2`	`-1`

Boolean Operators		
Operator	**Example**	**Result**
Equality	`1 == 2`	
Inequality	`1 != 2`	
Less Than	`1 < 2`	
Less Than or Equal To	`1 <= 2`	
Greater Than	`1 > 2`	
Greater Than or Equal To	`1 >= 2`	

New Type: Booleans

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

Remember

Numerical Operators		
Operator	**Example**	**Result**
Sum	`1 + 2`	`3`
Difference	`1 - 2`	`-1`

Boolean Operators		
Operator	**Example**	**Result**
Equality	`1 == 2`	`False`
Inequality	`1 != 2`	`True`
Less Than	`1 < 2`	`True`
Less Than or Equal To	`1 <= 2`	`True`
Greater Than	`1 > 2`	`False`
Greater Than or Equal To	`1 >= 2`	`False`

New Type: Booleans

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

Boolean Operators		
Operator	**Example**	**Result**
Equality	`1 == 2`	`False`
Inequality	`1 != 2`	`True`
Less Than	`1 < 2`	`True`
Less Than or Equal To	`1 <= 2`	`True`
Greater Than	`1 > 2`	`False`
Greater Than or Equal To	`1 >= 2`	`False`

Boolean Types

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

Boolean Operators			
Operator	**Examples**		**Result**
`and`	`(4<5) and (6<3)`		
`or`	`(4<5) or (6<3)`		
`not`	`not(4<5)`		

Boolean Types

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

Boolean Operators			
Operator	**Examples**		**Result**
`and`	`(4<5) and (6<3)`	`True and False`	
`or`	`(4<5) or (6<3)`	`True or False`	
`not`	`not(4<5)`	`not(True)`	

Boolean Types

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

Boolean Operators			
Operator	**Examples**		**Result**
`and`	`(4<5) and (6<3)`	`True and False`	`False`
`or`	`(4<5) or (6<3)`	`True or False`	`True`
`not`	`not(4<5)`	`not(True)`	`False`

Boolean Types

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

Boolean Operators			
Operator	**Examples**		**Result**
`and`	`(4<5) and (6<3)`	`True and False`	`False`
`or`	`(4<5) or (6<3)`	`True or False`	`True`
`not`	`not(4<5)`	`not(True)`	`False`

More Examples		**Result**
`(4<5) and ((6<3) or (5==5))`		
`(5==4) or (not(6<3))`		

Boolean Types

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

Boolean Operators			
Operator	**Examples**		**Result**
`and`	`(4<5) and (6<3)`	`True and False`	`False`
`or`	`(4<5) or (6<3)`	`True or False`	`True`
`not`	`not(4<5)`	`not(True)`	`False`

More Examples		**Result**
`(4<5) and ((6<3) or (5==5))`	`True and (False or True)`	
`(5==4) or (not(6<3))`		

Boolean Types

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

Boolean Operators			
Operator	**Examples**		**Result**
`and`	`(4<5) and (6<3)`	`True and False`	`False`
`or`	`(4<5) or (6<3)`	`True or False`	`True`
`not`	`not(4<5)`	`not(True)`	`False`

More Examples		**Result**
`(4<5) and ((6<3) or (5==5))`	`True and (False or True)`	`True`
`(5==4) or (not(6<3))`		

Boolean Types

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

Boolean Operators			
Operator	**Examples**		**Result**
`and`	`(4<5) and (6<3)`	`True and False`	`False`
`or`	`(4<5) or (6<3)`	`True or False`	`True`
`not`	`not(4<5)`	`not(True)`	`False`

More Examples		**Result**
`(4<5) and ((6<3) or (5==5))`	`True and (False or True)`	`True`
`(5==4) or (not(6<3))`	`False or not(False)`	

Boolean Types

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

Boolean Operators			
Operator	**Examples**		**Result**
`and`	`(4<5) and (6<3)`	`True and False`	`False`
`or`	`(4<5) or (6<3)`	`True or False`	`True`
`not`	`not(4<5)`	`not(True)`	`False`

More Examples		**Result**
`(4<5) and ((6<3) or (5==5))`	`True and (False or True)`	`True`
`(5==4) or (not(6<3))`	`False or not(False)`	`True`

Boolean Expressions on Strings

Boolean Operators on Strings		
Operator	**Example**	**Result**
Equality	`'a' == 'b'`	`False`
Inequality	`'a' != 'b'`	`True`
Less Than	`'a' < 'b'`	`True`
Less Than or Equal To	`'a' <= 'b'`	`True`
Greater Than	`'a' > 'b'`	`False`
Greater Than or Equal To	`'a' >= 'b'`	`False`

Review: Statements

- Expression Statements
- Assignment Statements
- List-Assignment Statements
- `For` Statements
- `If` Statements

Calculates something

Stores a value for a variable in memory table

Replaces
An item or slices of an existing list with new value(s)

"For each element in `myList`, do something"

If `A` is true, then do something, otherwise do something else

Boolean Statements (`If` Stmts)

- "If something's true, do A"

```
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"

```
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"

```
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"

```
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**)

```
[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)

```
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		
`len`	List	Integer
`len`	String	Integer
`len`	...	Integer

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

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

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

Python Functions

Preloaded Functions		
`len`	List **OR String**	Integer

Python Functions

Preloaded Functions		
`len`	List OR String OR ...	Integer
`float`	Number (as an Integer, Float, or String)	Float
`int`	Number (as an Integer, Float, or String)	Integer
`str`	Integer, Float, String, or List	String

These functions *cast* a variable of one type to another type

Python Functions

Preloaded Functions		
`len`	List OR String OR ...	Integer
`float`	Number (as an Integer, Float, or String)	Float
`int`	Number (as an Integer, Float, or String)	Integer
`str`	Integer, Float, String, or List	String
`range`	Two Integers 1. Start Index (Inclusive) 2. End Index (Exclusive)	List of Integers

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
```

Get the Longest Word in Moby Dick

```
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

```
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: 0

word: cat

```
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

```
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: 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: 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
```

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/ len(myList)
    return avg
```

How it works

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

return this

s: 16

avg: 4.0

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
```

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

```
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

```
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:

word: cat

```
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: cat

word: cat

```
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: 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"]

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"]

longestWord: elephant

word: zebra

```
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
```

How it works

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

longestWord: flying squirrel

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
```

How it works

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

return this

longestWord: flying squirrel

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
```

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/ 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

Function	Example
`addTwo(x,y)`	`addTwo(2,0)`
`subtractTwo(x,y)`	`subtractTwo(2,0)`
`multiplyTwo(x,y)`	`z = multiply(2,0)`
`divideTwo(x,y)`	`divideTwo(2,0)`
`addList(myList)`	`myList([2,0])`

Debugging Programs

Function	Example
`addTwo(x,y)`	`addTwo(2,0)` – output is wrong
`subtractTwo(x,y)`	`subtractTwo(2,0)` – any input causes an error
`multiplyTwo(x,y)`	`z = multiply(2,0)` – What is z after this assignment?
`divideTwo(x,y)`	`divideTwo(2,0)` – What happens when I run this? Use an "if" to catch the error and print a message to the screen.
`addList(myList)`	`myList([2,0])` – any input causes an error

Boolean Expressions on Numbers

Expression	Evaluation	Final
`(100 > 101) and (-1 != -1)`		
`(1 <= 2) or ((1 == 1) and (1 != 2)`		
`x = 1` `not(x > 2) or (x <= x+1)`		
`y = 100` `not(not(y == 100))`		

Boolean Expressions on Numbers

Expression	Evaluation	Final
`(100 > 101) and (-1 != -1)`	`False and False`	`False`
`(1 <= 2) or ((1 == 1) and (1 != 2)`		
`x = 1` `not(x > 2) or (x <= x+1)`		
`y = 100` `not(not(y == 100))`		

Boolean Expressions on Numbers

Expression	Evaluation	Final
`(100 > 101) and (-1 != -1)`	`False and False`	`False`
`(1 <= 2) or ((1 == 1) and (1 != 2)`	`True or (True and True)`	`True`
`x = 1` `not(x > 2) or (x <= x+1)`		
`y = 100` `not(not(y == 100))`		

Boolean Expressions on Numbers

Expression	Evaluation	Final
`(100 > 101) and (-1 != -1)`	`False and False`	`False`
`(1 <= 2) or ((1 == 1) and (1 != 2)`	`True or (True and True)`	`True`
`x = 1` `not(x > 2) or (x <= x+1)`	`True or True`	`True`
`y = 100` `not(not(y == 100))`		

Boolean Expressions on Numbers

Expression	Evaluation	Final
`(100 > 101) and (-1 != -1)`	`False and False`	`False`
`(1 <= 2) or ((1 == 1) and (1 != 2)`	`True or (True and True)`	`True`
`x = 1` `not(x > 2) or (x <= x+1)`	`True or True`	`True`
`y = 100` `not(not(y == 100))`	`not(False)`	`True`

Boolean Expressions on Strings

Boolean Operators on Strings		
Operator	**Example**	**Result**
Equality	`'a' == 'b'`	`False`
Inequality	`'a' != 'b'`	`True`

Boolean Expressions on Strings

Boolean Operators on Strings		
Operator	**Example**	**Result**
Equality	`'a' == 'b'`	`False`
Inequality	`'a' != 'b'`	`True`

```
>>> '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

```
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

```
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.'''
```

```
def testNoReplicates():
```

```
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

Preloaded Functions			
Name	**Inputs**	**Outputs**	**CHANGES**
`sort`	List		Original List!

Sorting Lists

Preloaded Functions			
Name	**Inputs**	**Outputs**	**CHANGES**
`sort`	List		Original List!

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

Sorting Lists

Preloaded Functions			
Name	**Inputs**	**Outputs**	**CHANGES**
`sort`	List		Original List!

```
>>> 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