

# Vocabulary Size of Moby Dick

March 8, 2012

# 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

# Boolean Expressions on Numbers

Expression	Evaluation	Final
<code>(100 &gt; 101) and (-1 != -1)</code>		
<code>(1 &lt;= 2) or ((1 == 1) and (1 != 2))</code>		
<code>x = 1</code> <code>not(x &gt; 2) or (x &lt;= x+1)</code>		
<code>y = 100</code> <code>not(not(y == 100))</code>		

# Boolean Expressions on Numbers

Expression	Evaluation	Final
<code>(100 &gt; 101) and (-1 != -1)</code>	False and False	False
<code>(1 &lt;= 2) or ((1 == 1) and (1 != 2))</code>		
<code>x = 1</code> <code>not(x &gt; 2) or (x &lt;= x+1)</code>		
<code>y = 100</code> <code>not(not(y == 100))</code>		

# Boolean Expressions on Numbers

Expression	Evaluation	Final
<code>(100 &gt; 101) and (-1 != -1)</code>	False and False	False
<code>(1 &lt;= 2) or ((1 == 1) and (1 != 2))</code>	True or (True and True)	True
<code>x = 1</code> <code>not(x &gt; 2) or (x &lt;= x+1)</code>		
<code>y = 100</code> <code>not(not(y == 100))</code>		

# Boolean Expressions on Numbers

Expression	Evaluation	Final
<code>(100 &gt; 101) and (-1 != -1)</code>	False and False	False
<code>(1 &lt;= 2) or ((1 == 1) and (1 != 2))</code>	True or (True and True)	True
<code>x = 1</code> <code>not(x &gt; 2) or (x &lt;= x+1)</code>	True or True	True
<code>y = 100</code> <code>not(not(y == 100))</code>		

# Boolean Expressions on Numbers

Expression	Evaluation	Final
<code>(100 &gt; 101) and (-1 != -1)</code>	False and False	False
<code>(1 &lt;= 2) or ((1 == 1) and (1 != 2))</code>	True or (True and True)	True
<code>x = 1</code> <code>not(x &gt; 2) or (x &lt;= x+1)</code>	True or True	True
<code>y = 100</code> <code>not(not(y == 100))</code>	not(False)	True

# Boolean Expressions on Strings

Boolean Operators on Strings		
Operator	Example	Result
Equality	<code>'a' == 'b'</code>	False
Inequality	<code>'a' != 'b'</code>	True

# Boolean Expressions on Strings

Boolean Operators on Strings		
Operator	Example	Result
Equality	<code>'a' == 'b'</code>	False
Inequality	<code>'a' != 'b'</code>	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 testNoReplicates():
```

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

```
def testIsElementOf():
```

Writing *test cases* are important to make sure your program works!

# Break

- Look at *Fusion Tables* (url on website)
  - Created by **Jayant Madhavan** at Google Research



The image shows a screenshot of the MTBGuru website. On the left, there is a table titled "MTBGuru tracks" with columns for "Track #", "Name", "Map URL", "Distance", and "Elevation". A large blue arrow points from the table to a map on the right. The map shows a geographical area with red lines representing tracks. A vertical line on the map indicates an elevation profile for a specific track. The map interface includes a search bar, a "Map" button, and a "Google Earth" button. On the right side of the map, there is a sidebar with various links and a "Google Earth" button.

Track #	Name	Map URL	Distance	Elevation
1	Montana de Oro	<a href="http://www.mtbguru.com/track/1">http://www.mtbguru.com/track/1</a>	388.0	1438.00
2	Fast One	<a href="http://www.mtbguru.com/track/2">http://www.mtbguru.com/track/2</a>	492.800	812.700
3	Black Range	<a href="http://www.mtbguru.com/track/3">http://www.mtbguru.com/track/3</a>	620.857	985.000
4	Black Diamond	<a href="http://www.mtbguru.com/track/4">http://www.mtbguru.com/track/4</a>	750.000	750.000
5	Kudon Trail in English	<a href="http://www.mtbguru.com/track/5">http://www.mtbguru.com/track/5</a>	755.000	1238.000
6	Golden Canyon	<a href="http://www.mtbguru.com/track/6">http://www.mtbguru.com/track/6</a>	760.000	580.000
7	Rocky Trail	<a href="http://www.mtbguru.com/track/7">http://www.mtbguru.com/track/7</a>	750.000	750.000
8	Sanjo Taurus spring hill	<a href="http://www.mtbguru.com/track/8">http://www.mtbguru.com/track/8</a>	450.000	450.000
9	Yosemite Trail	<a href="http://www.mtbguru.com/track/9">http://www.mtbguru.com/track/9</a>	400.000	400.000
10	Rocking Canyon	<a href="http://www.mtbguru.com/track/10">http://www.mtbguru.com/track/10</a>	450.000	450.000
11	South Mountain Phoenix	<a href="http://www.mtbguru.com/track/11">http://www.mtbguru.com/track/11</a>	750.000	750.000
12	Yosemite Valley to	<a href="http://www.mtbguru.com/track/12">http://www.mtbguru.com/track/12</a>	750.000	750.000
13	Yosemite Valley to	<a href="http://www.mtbguru.com/track/13">http://www.mtbguru.com/track/13</a>	750.000	750.000
14	Yosemite Valley to	<a href="http://www.mtbguru.com/track/14">http://www.mtbguru.com/track/14</a>	750.000	750.000
15	Yosemite Valley to	<a href="http://www.mtbguru.com/track/15">http://www.mtbguru.com/track/15</a>	750.000	750.000
16	Yosemite Valley to	<a href="http://www.mtbguru.com/track/16">http://www.mtbguru.com/track/16</a>	750.000	750.000
17	Yosemite Valley to	<a href="http://www.mtbguru.com/track/17">http://www.mtbguru.com/track/17</a>	750.000	750.000
18	Yosemite Valley to	<a href="http://www.mtbguru.com/track/18">http://www.mtbguru.com/track/18</a>	750.000	750.000
19	Yosemite Valley to	<a href="http://www.mtbguru.com/track/19">http://www.mtbguru.com/track/19</a>	750.000	750.000
20	Yosemite Valley to	<a href="http://www.mtbguru.com/track/20">http://www.mtbguru.com/track/20</a>	750.000	750.000

# 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
<code>sort</code>	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']
```

# Python `For` Statements (For Loops)

“For each element in list `myList`, do something”

```
>>> myList = [1,2,3]
>>> for i in range(0,3):
    print myList[i]

1
2
3
>>>
```

# Python For Statements (For Loops)

“For each element in list myList, do something”

```
>>> myList = [1,2,3]
>>> for i in range(0,3):
    print myList[i]
```

```
1
```

```
2
```

```
3
```

```
>>>
```



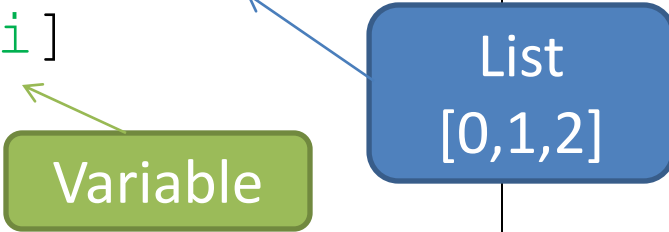
List  
[0,1,2]

# Python For Statements (For Loops)

“For each element in list myList, do something”

```
>>> myList = [1,2,3]
>>> for i in range(0,3):
    print myList[i]
```

1  
2  
3  
>>>



The diagram consists of two callout boxes. A blue rounded rectangle on the right contains the text "List [0,1,2]". A blue arrow points from this box to the "range(0,3)" part of the code. A green rounded rectangle below it contains the text "Variable". A green arrow points from this box to the variable "i" in the code.

# Python For Statements (For Loops)

“For each element in list myList, do something”

```
>>> myList = [1,2,3]
>>> for i in range(0,3):
    print myList[i]
```

1  
2  
3  
>>>

The diagram illustrates the execution of a Python for loop. It shows a code block with three lines of code: `myList = [1,2,3]`, `for i in range(0,3):`, and `print myList[i]`. The code is annotated with a blue box labeled "List [0,1,2]" pointing to the `range(0,3)` function, and a green box labeled "Variable" pointing to the variable `i`. Below the code, a rounded rectangle contains the question: "Q: What if we don't know the length of the list?". To the left of the code, the numbers 1, 2, and 3 are listed vertically, followed by `>>>`, indicating the output of the loop.

# Python For Statements (For Loops)

“For each element in list myList, do something”

```
>>> myList = [1,2,3]
>>> for i in range(0, len(myList)):
    print myList[i]
```

1  
2  
3  
>>>

Q: What if we don't know the length of the list?

# 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
- Practice **debugging** functions

# Debugging Programs

Download `functionErrors.py` & open it in IDLE. Each function has an error or takes an input that results in an error.

Function	Example
<code>addTwo(x, y)</code>	<code>addTwo(1, 2)</code> – output is wrong
<code>subtractTwo(x, y)</code>	any inputs cause an error
<code>multiplyTwo(x, y)</code>	<code>z = multiplyTwo(x, y)</code> – what is z after?
<code>divideTwo(x, y)</code>	<code>divideTwo(2, 0)</code> – write an if statement to “catch” this problem and print a message to the screen.
<code>addList(myList)</code>	any inputs cause an error