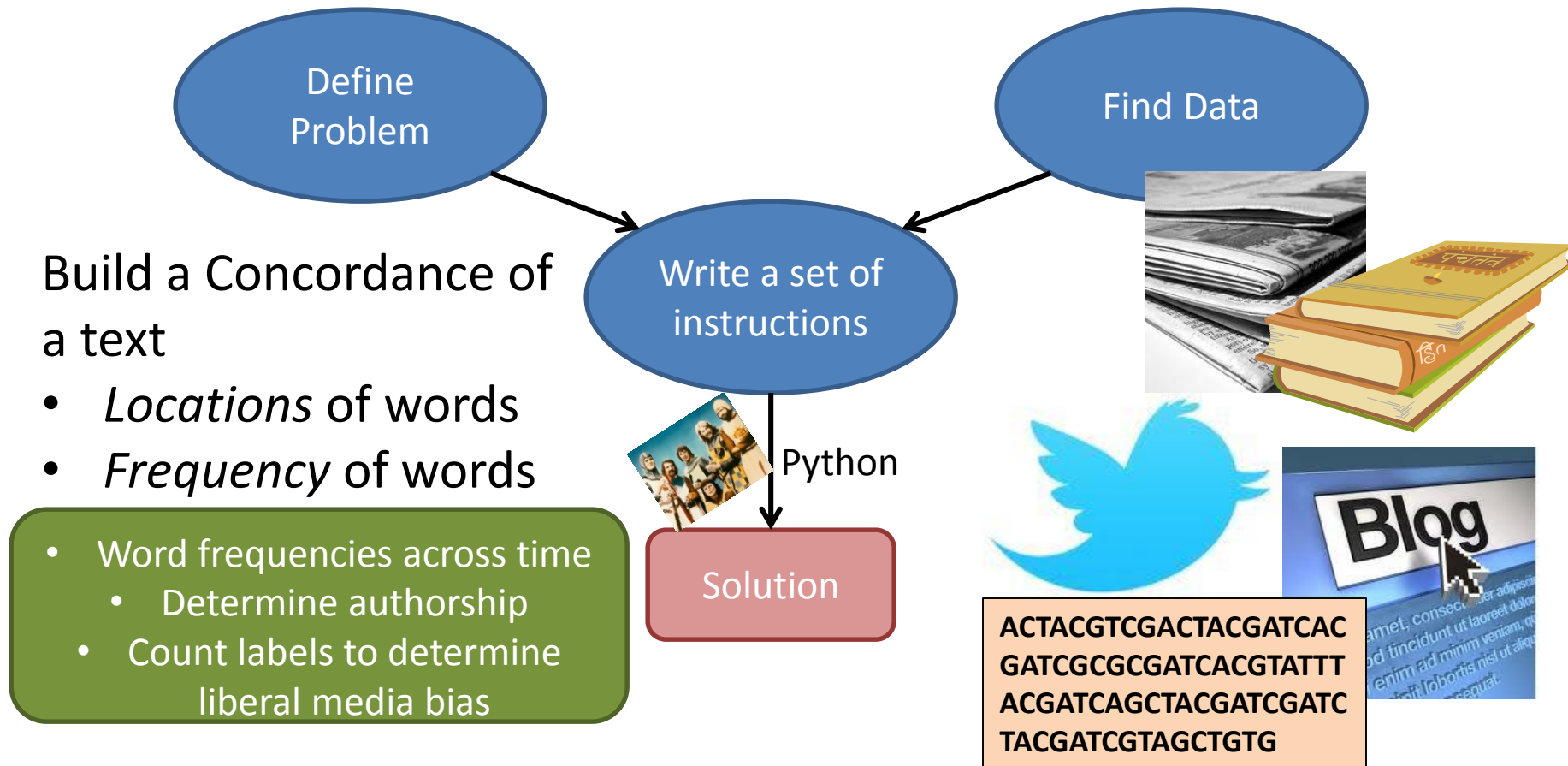


Textual Analysis: Summary Statistics, part 1

Mar 1, 2016

Textual Analysis



The Big Picture

Overall Goal

Build a Concordance of a text

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

Today: Summary Statistics

- Review material from last class
- Learn how to read in a text file and create a list of words
- Count the number of words in poem.txt (by Shel Silverstein)
- Count the number of words in Moby Dick
- Compute the average word length of Moby Dick
- Find the longest word in Moby Dick

ACT2-1, Task 1 from last class

- You wrote the `addOne` function
- Goal: takes an input, `t`; computes `t+1` and returns it

Question: What is the difference between *returning* a value and *printing* a value?

```
def addOne(t):  
    y = t + 1  
    return y
```

vs.

```
def addOne(t):  
    y = t + 1  
    print(y)  
    return
```

ACT2-1

```
def addOne(t):  
    y = t + 1  
    return y
```

vs.

```
def addOne(t):  
    y = t + 1  
    print(y)  
    return
```

```
>>> four = addOne(3)
```

```
>>> four = addOne(3)
```

ACT2-1

```
def addOne(t):  
    y = t + 1  
    return y
```

vs.

```
def addOne(t):  
    y = t + 1  
    print(y)  
    return
```

```
>>> four = addOne(3)  
>>>
```

```
>>> four = addOne(3)  
4  
>>>
```

ACT2-1

```
def addOne(t):  
    y = t + 1  
    return y
```

vs.

```
def addOne(t):  
    y = t + 1  
    print(y)  
    return
```

```
>>> four = addOne(3)  
>>> four
```

```
>>> four = addOne(3)  
4  
>>> four
```

ACT2-1

```
def addOne(t):  
    y = t + 1  
    return y
```

vs.

```
def addOne(t):  
    y = t + 1  
    print(y)  
    return
```

```
>>> four = addOne(3)  
>>> four  
4  
>>>
```

```
>>> four = addOne(3)  
4  
>>> four  
>>>
```

ACT2-1

```
def addOne(t):  
    y = t + 1  
    return y
```

vs.

```
def addOne(t):  
    y = t + 1  
    print(y)  
    return
```

```
>>> four = addOne(3)  
>>> four  
4  
>>>
```

```
>>> four = addOne(3)  
4  
>>> four  
>>>
```

addOne(3) returned “nothing”! So four was assigned a “nothing” value

ACT2-1

```
def addOne(t):  
    y = t + 1  
    return y
```

vs.

```
def addOne(t):  
    y = t + 1  
    print(y)  
    return
```

```
>>> four = addOne(3)  
>>> four  
4  
>>>
```

```
>>> four = addOne(3)  
4  
>>> four  
>>> type(four)  
<type 'NoneType'>
```

addOne(3) returned “nothing”! So four was assigned a “nothing” value

ACT2-2

- Try a new function called `split()`
- `split()` returns a list of words in a string separated by whitespace or a specified delimiter

- You'd expect

```
split(myString)
```

- But actually called “on” on a string object:

```
myString.split()
```

ACT2-2, preparation for task 1

- A new syntax: `<var>.<function>`
- Example: `myString.split()`
- Names a function associated with a particular type of variables
- While `print(...)` works for any type, `split()` works only for strings...
 - So it gets this special form, with the string name as prefix

ACT2-2

```
myString.split()
```



Empty, or a substring that will
act as a delimiter

ACT2-2

- In Python, some functions are only defined for certain kinds of objects, others aren't:

`myString.split()` versus
`type()` or `our addOne()`

- Takes practice and patience to learn which functions are written this way
- There's no rule about which form to use, so no *consistent* pattern of use

ACT2-2

- Do Task 1

ACT2-2, Task 2 preparation

- Filenames
 - String with special characters
- File – a new type
- Handling files – brief introduction

String reminders

Escape Characters

- `'\'` means interpret the NEXT character differently.
- `\n` : “new line”
 - `\'` : “apostrophe”
 - `\t` : “tab”

Escape Characters

`'\'` means interpret the NEXT character differently in Python, so in order to treat the `'\'` just as a regular character, you double it (`\\`).

Filenames

- Files on your computer have names like

`C:\Users\Alexandra\Downloads\poem.txt`

- In Python, names are represented by strings
- Remember that “\” is a special character, so “\t” means “tab character”
- To put filename in a string, use “\\” to get “\”:

```
>>> fileName = "C:\\Users\\Alexandra\\Downloads\\poem.txt"
```

Working with Files

“Inputs” are also called *Arguments*.

Preloaded Functions

Name	Inputs	Outputs
open	Two Strings 1. File Name 2. “r” for read (for now)	File
read (On a File)	none	String
close (On a File)	none	none
split (On a String)	(optional) delimiter	List of Strings

Working with Files

1. Save `poem.txt` from the webpage.
2. Right-click and select 'Properties' / 'Get Info'
3. Note the file location
(C:\Users\Alexandra\Downloads...)
4. In Python, write an assignment statement that stores the file location as a string.

```
>>> fileName = "C:\\Users\\Alexandra\\Downloads\\poem.txt"
```

Working with Files

Preloaded Functions		
Name	Inputs	Outputs
type	Expression	Type
open	Two Strings 1. File Name 2. "r" for read (for now)	File

File is a NEW Type

```
>>> fileName = "C:\\Users\\Alexandra\\Downloads\\poem.txt"  
>>>
```

Working with Files

Preloaded Functions

Name	Inputs	Outputs
type	Expression	Type
open	Two Strings 1. File Name 2. "r" for read (for now)	File

File is a NEW Type

```
>>> fileName = "C:\\Users\\Alexandra\\Downloads\\poem.txt"
>>> myFile = open(fileName, "r")
```

Working with Files

Preloaded Functions		
Name	Inputs	Outputs
type	Expression	Type
open	Two Strings 1. File Name 2. "r" for read (for now)	File
read (On a File)	none	String

```
>>> fileName = "C:\\Users\\Alexandra\\Downloads\\poem.txt"
>>> myFile = open(fileName, "r")
```

Working with Files

Preloaded Functions		
Name	Inputs	Outputs
type	Expression	Type
open	Two Strings 1. File Name 2. "r" for read (for now)	File
read (On a File)	none	String

```
>>> fileName = "C:\\Users\\Alexandra\\Downloads\\poem.txt"
>>> myFile = open(fileName, "r")
>>> fileString = myFile.read()
```

Working with Files

Preloaded Functions		
Name	Inputs	Outputs
type	Expression	Type
open	Two Strings 1. File Name 2. "r" for read (for now)	File
read (On a File)	none	String
close (On a File)	none	none

```
>>> fileName = "C:\\Users\\Alexandra\\Downloads\\poem.txt"
>>> myFile = open(fileName, "r")
>>> fileString = myFile.read()
>>> myFile.close()
```

Working with Files

```
>>> shellList = readShel()
>>> shellList
['Sarah', 'Cynthia', 'Sylvia', 'Stout', 'Would', 'not',
'take', 'the', 'garbage', 'out!', "She'd", 'scour', 'the',
'pots', 'and', 'scrape', 'the', 'pans,', 'Candy', 'the',
'yams', 'and', 'spice', 'the', 'hams,', 'And', 'though',
'her', 'daddy', 'would', 'scream', 'and', 'shout,', 'She',
'simply', 'would', 'not', 'take', 'the', 'garbage',
'out.', 'And', 'so', 'it', 'piled', 'up', 'to', 'the',
'ceilings:', 'Coffee', 'grounds,', 'potato', 'peelings,',
'Brown',
...
'an', 'awful', 'fate,', 'That', 'I', 'cannot', 'now',
'relate', 'Because', 'the', 'hour', 'is', 'much', 'too',
'late.', 'But', 'children,', 'remember', 'Sarah', 'Stout',
'And', 'always', 'take', 'the', 'garbage', 'out!']
```

Working with Files

```
>>> shellList = readShel()
>>> shellList
['Sarah', 'Cynthia', 'Sylvia', 'Stout', 'Would', 'not',
'take', 'the', 'garbage', 'out!', 'And', 'always', 'take', 'the', 'garbage', 'out!'],
'pot', 'yam', 'her', 'sim', 'out', 'cei', 'Bro',
...
'an', 'awful', 'fate,', 'That', 'I', 'cannot', 'now',
'relate', 'Because', 'the', 'hour', 'is', 'much', 'too',
'late.', 'But', 'children,', 'remember', 'Sarah', 'Stout',
'And', 'always', 'take', 'the', 'garbage', 'out!']
```

Escape Characters

\\ means interpret the NEXT character differently.

- \n: “new line”
- \' : “apostrophe”
- \t: “tab”

ACT2-2

- Do Task 2

Python `For` Statements (For Loops)

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

```
>>> myList = [1,2,3]
>>>
```

Python `For` Statements (For Loops)

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

```
>>> myList = [1,2,3]
>>> for element in myList:
...     print element

1
2
3
>>>
```

Python For Statements (For Loops)

“For each element in list myList, do something”

```
>>> myList = [1,2,3]
>>> for element in myList:
...     print element
```

1

2

3

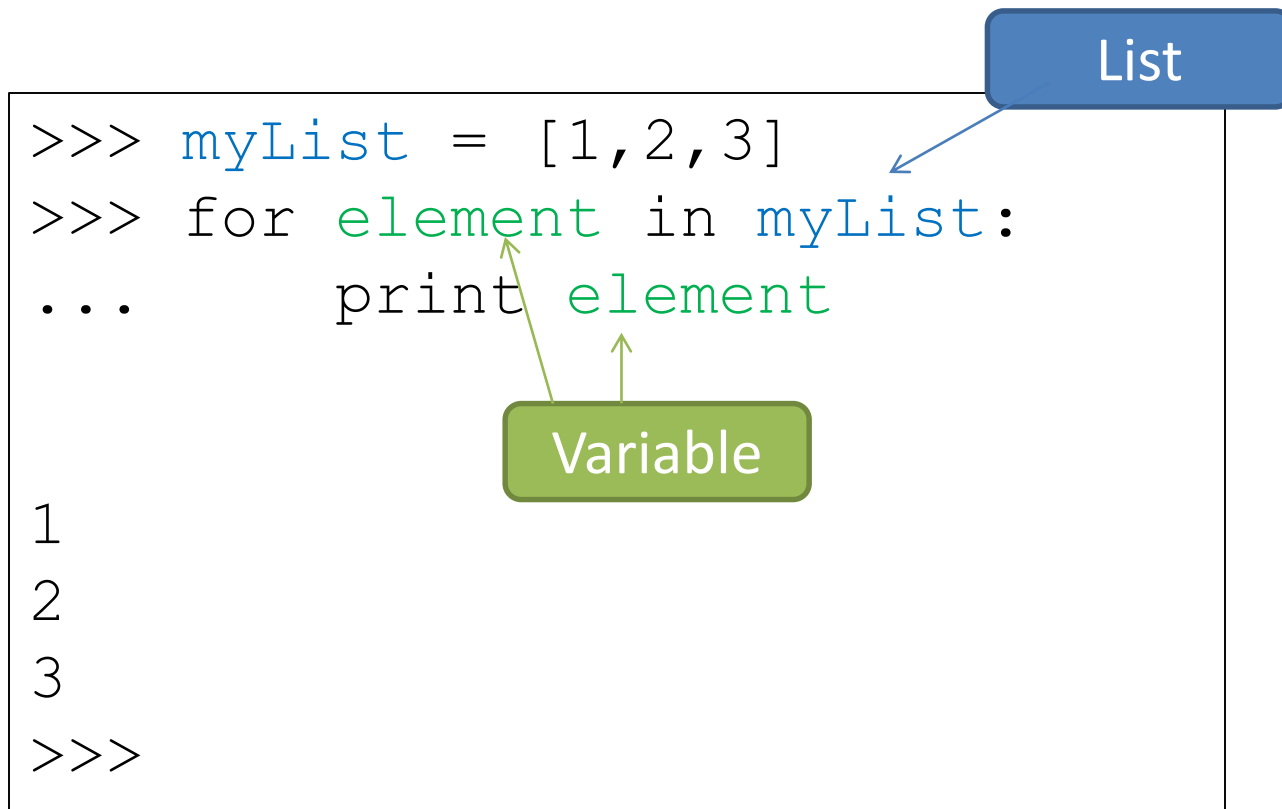
```
>>>
```

List



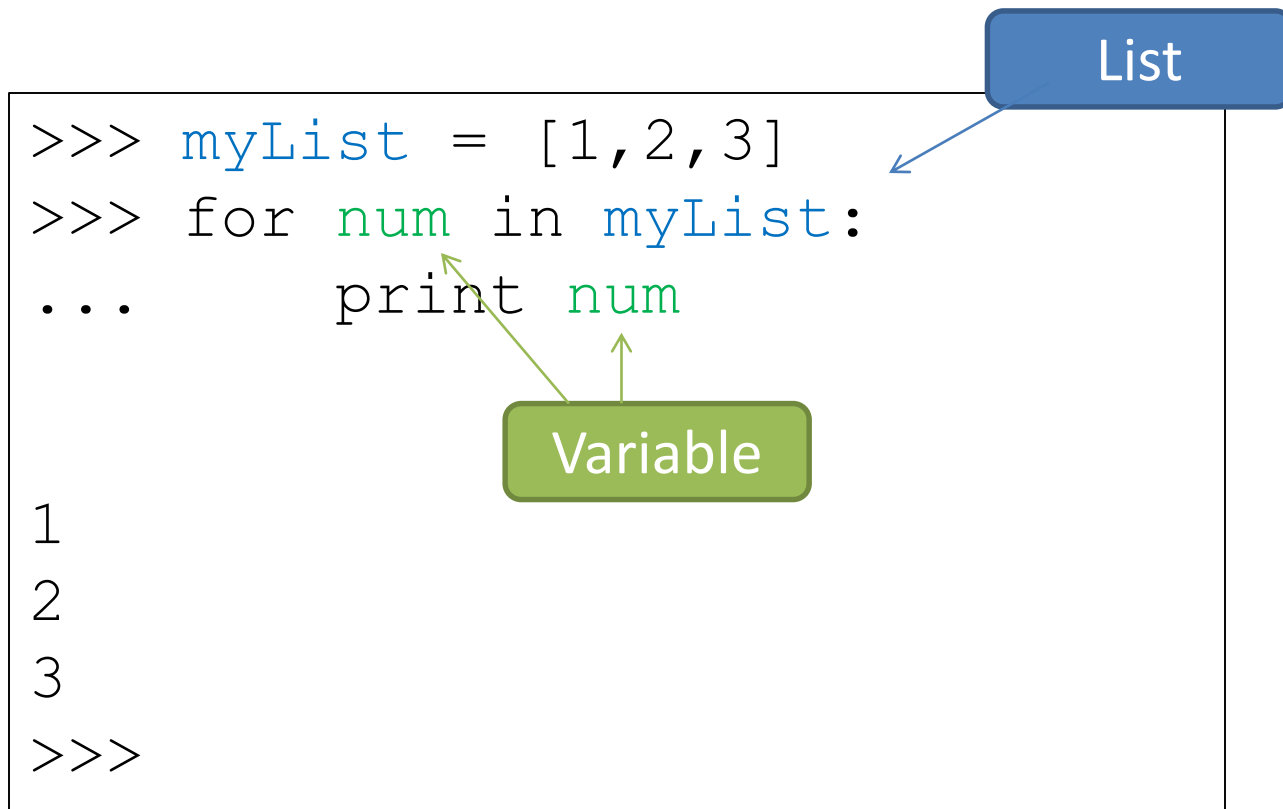
Python For Statements (For Loops)

“For each element in list myList, do something”



Python For Statements (For Loops)

“For each element in list myList, do something”



Python For Statements (For Loops)

“For each element in list myList, do something”

```
>>> myList = [1,2,3]
>>> for num in myList:
...     print num
```

1
2
3
>>>

The diagram shows a Python code snippet with three annotations:

- A blue box labeled "List" with an arrow pointing to the `myList` variable in the `for` statement.
- A green box labeled "Variable" with an arrow pointing to the `num` variable in the `print` statement.
- A red box labeled "Indentation Matters!!" with a bracket pointing to the indentation of the `print num` line.

Activity 2-2

- Do Task 3

Word Count for Shel's Poem

```
def countWordsInShel():  
    '''Returns the number of words in the poem.'''  
  
    return count
```

Word Count for Shel's Poem

```
def countWordsInShel():  
    '''Returns the number of words in the poem.'''  
    myList = readShel()  
    # the 'count' variable counts the number of words  
    count = 0  
    for word in myList:  
        count = count + 1  
    print "There are ",count," words in the poem."  
    return count
```

Word Count for Shel's Poem

Good Programming Practices: Documentation!

```
def countWordsInShel():  
    '''Returns the number of words in the poem.'''  
    myList = readShel()  
    # the 'count' variable counts the number of words  
    count = 0  
    for word in myList:  
        count = count + 1  
    print "There are ", count, " words in the poem."  
    return count
```

Program Description
(triple quotes)

Comment (#)

Print Statement

Execution model for “for” loops

- If the loop variable isn't in the memory table...add it
- Repeatedly assign to it sequential items in the list...
- ...and execute the statements within the loop
- Note: when done, the loop variable will be in the memory table, with its last value

The Big Picture

Overall Goal

Build a Concordance of a text

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

Today: Summary Statistics

- Review material from last class
- Count the number of words in poem.txt (by Shel Silverstein)
- Count the number of words in Moby Dick
 - There's a shortcut...
- Compute the average word length of Moby Dick
- Find the longest word in Moby Dick

A Shortcut to List Length

Preloaded Functions

len	List	Integer
-----	------	---------

```
>>> len(myList)
```

A Shortcut to List Length

Preloaded Functions

len

List

Integer

```
>>> len(myList)
```

Today: Summary Statistics

- Review material from last class
- Count the number of words in poem.txt (by Shel Silverstein)
- Count the number of words in Moby Dick
 - There's a shortcut...
- Compute the average word length of Moby Dick
- Find the longest word in Moby Dick

Python Functions

Preloaded Functions

len	List OR String	Integer
-----	----------------	---------

Python Functions

Preloaded Functions		
<code>len</code>	List OR String	Integer
<code>float</code>	Number (as an Integer, Float, or String)	Float
<code>int</code>	Number (as an Integer, Float, or String)	Integer
<code>str</code>	Integer, Float, String, or List	String

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

- `3 / 4 -> 0.75`
- `3/float(4) -> 0.75`, `float(3)/4 -> 0.75`, `float(3)/float(4) -> 0.75`
- if an arithmetic expression involves a float, the result will be a float. `3 + 0.0 -> 3.0`, `3 + float(0) -> 3.0`
- New shorthand: “->” means “evaluates to”

Python Functions

Preloaded Functions

len	List OR String	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



Activity

- Do Task 4

Compute the Average Word Length of Moby Dick

```
def avgWordLengthInMobyDick():  
    '''Gets the average word length in MobyDick.txt'''
```

Compute the Average Word Length of Moby Dick

```
def avgWordLengthInMobyDick():  
    '''Gets the average word length in MobyDick.txt'''  
    myList = readMobyDick()  
    s = 0  
    for word in myList:  
        s = s + len(word)  
    avg = s / len(myList)  
    return avg
```

Is our Program Correct?

```
>>> MDList = readMobyDick()
>>> MDList[0:99]
['CHAPTER', '1', 'Loomings', 'Call', 'me', 'Ishmael.',
'Some', 'years', 'ago--never', 'mind', 'how', 'long',
'precisely--', 'having', 'little', 'or', 'no', 'money', 'in',
'my', 'purse,', 'and', 'nothing', 'particular', 'to',
'interest', 'me', 'on', 'shore,', 'I', 'thought', 'I',
'would', 'sail', 'about', 'a', 'little', 'and', 'see', 'the',
'watery', 'part', 'of', 'the', 'world.', 'It', 'is', 'a',
'way', 'I', 'have', 'of', 'driving', 'off', 'the', 'spleen',
'and', 'regulating', 'the', 'circulation.', 'Whenever', 'I',
'find', 'myself', 'growing', 'grim', 'about', 'the',
'mouth;', 'whenever', 'it', 'is', 'a', 'damp,', 'drizzly',
'November', 'in', 'my', 'soul;', 'whenever', 'I', 'find',
'myself', 'involuntarily', 'pausing', 'before', 'coffin',
'warehouses,', 'and', 'bringing', 'up', 'the', 'rear', 'of',
'every', 'funeral', 'I', 'meet;', 'and']
```

The Big Picture

Overall Goal

Build a Concordance of a text

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

Today: Summary Statistics

- Review material from last class
- Count the number of words in poem.txt (by Shel Silverstein)
- Count the number of words in Moby Dick
 - There's a shortcut...
- Compute the average word length of Moby Dick
- Find the longest word in Moby Dick

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	<code>1 + 2</code>	3
Difference	<code>1 - 2</code>	-1

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

New Type: Booleans

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

Remember

Numerical Operators		
Operator	Example	Result
Sum	<code>1 + 2</code>	<code>3</code>
Difference	<code>1 - 2</code>	<code>-1</code>

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

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	<code>1 == 2</code>	<code>False</code>
Inequality	<code>1 != 2</code>	<code>True</code>
Less Than	<code>1 < 2</code>	<code>True</code>
Less Than or Equal To	<code>1 <= 2</code>	<code>True</code>
Greater Than	<code>1 > 2</code>	<code>False</code>
Greater Than or Equal To	<code>1 >= 2</code>	<code>False</code>

Boolean Types

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

Boolean Operators			
Operator	Examples		Result
<code>and</code>	<code>(4<5) and (6<3)</code>		
<code>or</code>	<code>(4<5) or (6<3)</code>		
<code>not</code>	<code>not (4<5)</code>		

Boolean Types

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

Boolean Operators			
Operator	Examples		Result
<code>and</code>	<code>(4<5) and (6<3)</code>	<code>True and False</code>	
<code>or</code>	<code>(4<5) or (6<3)</code>	<code>True or False</code>	
<code>not</code>	<code>not (4<5)</code>	<code>not (True)</code>	

Boolean Types

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

Boolean Operators			
Operator	Examples		Result
<code>and</code>	<code>(4<5) and (6<3)</code>	<code>True and False</code>	<code>False</code>
<code>or</code>	<code>(4<5) or (6<3)</code>	<code>True or False</code>	<code>True</code>
<code>not</code>	<code>not (4<5)</code>	<code>not (True)</code>	<code>False</code>

Boolean Types

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

Boolean Operators			
Operator	Examples		Result
<code>and</code>	<code>(4<5) and (6<3)</code>	<code>True and False</code>	<code>False</code>
<code>or</code>	<code>(4<5) or (6<3)</code>	<code>True or False</code>	<code>True</code>
<code>not</code>	<code>not (4<5)</code>	<code>not (True)</code>	<code>False</code>

More Examples		Result
<code>(4<5) and ((6<3) or (5==5))</code>		
<code>(5==4) or (not (6<3))</code>		

Boolean Types

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

Boolean Operators			
Operator	Examples		Result
<code>and</code>	<code>(4<5) and (6<3)</code>	<code>True and False</code>	<code>False</code>
<code>or</code>	<code>(4<5) or (6<3)</code>	<code>True or False</code>	<code>True</code>
<code>not</code>	<code>not (4<5)</code>	<code>not (True)</code>	<code>False</code>

More Examples		Result
<code>(4<5) and ((6<3) or (5==5))</code>	<code>True and (False or True)</code>	
<code>(5==4) or (not (6<3))</code>		

Boolean Types

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

Boolean Operators			
Operator	Examples		Result
<code>and</code>	<code>(4<5) and (6<3)</code>	<code>True and False</code>	<code>False</code>
<code>or</code>	<code>(4<5) or (6<3)</code>	<code>True or False</code>	<code>True</code>
<code>not</code>	<code>not (4<5)</code>	<code>not (True)</code>	<code>False</code>

More Examples		Result
<code>(4<5) and ((6<3) or (5==5))</code>	<code>True and (False or True)</code>	<code>True</code>
<code>(5==4) or (not (6<3))</code>		

Boolean Types

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

Boolean Operators			
Operator	Examples		Result
<code>and</code>	<code>(4<5) and (6<3)</code>	<code>True and False</code>	<code>False</code>
<code>or</code>	<code>(4<5) or (6<3)</code>	<code>True or False</code>	<code>True</code>
<code>not</code>	<code>not (4<5)</code>	<code>not (True)</code>	<code>False</code>

More Examples		Result
<code>(4<5) and ((6<3) or (5==5))</code>	<code>True and (False or True)</code>	<code>True</code>
<code>(5==4) or (not (6<3))</code>	<code>False or not (False)</code>	

Boolean Types

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

Boolean Operators			
Operator	Examples		Result
<code>and</code>	<code>(4<5) and (6<3)</code>	<code>True and False</code>	<code>False</code>
<code>or</code>	<code>(4<5) or (6<3)</code>	<code>True or False</code>	<code>True</code>
<code>not</code>	<code>not (4<5)</code>	<code>not (True)</code>	<code>False</code>

More Examples		Result
<code>(4<5) and ((6<3) or (5==5))</code>	<code>True and (False or True)</code>	<code>True</code>
<code>(5==4) or (not (6<3))</code>	<code>False or not (False)</code>	<code>True</code>

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)
```

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()  
    longestLen = 0  
    longestWord = ""  
    for word in myList:  
        if len(word) > longestLen:  
            longestLen = len(word)  
            longestWord = word  
    return longestWord
```

The Big Picture

Overall Goal

Build a Concordance of a text

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

Today: Summary Statistics

- Review material from last class
- Count the number of words in poem.txt (by Shel Silverstein)
- Count the number of words in Moby Dick
 - There's a shortcut...
- Compute the average word length of Moby Dick
- Find the longest word in Moby Dick

Is our Program Correct?

```
>>> shellList
['Sarah', 'Cynthia', 'Sylvia', 'Stout', 'Would', 'not',
'take', 'the', 'garbage', 'out!', "She'd", 'scour', 'the',
'pots', 'and', 'scrape', 'the', 'pans,', 'Candy', 'the',
'yams', 'and', 'spice', 'the', 'hams,', 'And', 'though',
'her', 'daddy', 'would', 'scream', 'and', 'shout,', 'She',
'simply', 'would', 'not', 'take', 'the', 'garbage',
'out.', 'And', 'so', 'it', 'piled', 'up', 'to', 'the',
'ceilings:', 'Coffee', 'grounds,', 'potato', 'peelings,',
'Brown',
...
'an', 'awful', 'fate,', 'That', 'I', 'cannot', 'now',
'relate', 'Because', 'the', 'hour', 'is', 'much', 'too',
'late.', 'But', 'children,', 'remember', 'Sarah', 'Stout',
'And', 'always', 'take', 'the', 'garbage', 'out!']
```

We'll come back to this...next class