Textual Analysis: Summary Statistics, part 1

February 26, 2015
Note

Getting TA help with Python: If you send email, be sure to cut and paste the error message and your code
Textual Analysis

Define Problem

Find Data

Write a set of instructions

Solution

Build a Concordance of a text
- **Locations** of words
- **Frequency** of words
- **Word frequencies across time**
  - Determine authorship
  - Count labels to determine liberal media bias

Python

ACTACGTCGACTACGATCA
CGATCGCGCGATCACGTAT
TTACGATCAGCTACGATCG
ATCTACGATCGTAGCTGTG
ATCG
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)
def addOne(t):
    y = t + 1
    return y

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

>>> four = addOne(3)
4
>>>
def addOne(t):
    y = t + 1
    return y

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

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

>>> four = addOne(3)
>>> four
def addOne(t):
    y = t + 1
    return y

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

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

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

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

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

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

addOne(3) returned “nothing”! So four was assigned a “nothing” value
```python
def addOne(t):
    y = t + 1
    return y
```

vs.

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

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

  ```python
  split(myString)
  ```

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

  ```python
  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
myString.split()
• In Python, some functions are only defined for certain kinds of objects, others aren’t:

\texttt{myString.split()} versus \texttt{type()} or our \texttt{addOne()}

• Takes practice and patience to learn which functions are written this way

• There’s no rule about which form to use, so no \textit{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
Escape Characters
\’ means interpret the NEXT character differently.
• \n: “new line”
• \’ : “apostrophe”
• \t : “tab”
Filenames

• Files on your computer have names like
  C:\Users\Steve\Desktop\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 “\”:

```python
>>> fileName = "C:\\Users\\Steve\\Desktop\\poem.txt"
```
## Working with Files

<table>
<thead>
<tr>
<th>Preloaded Functions</th>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
<td><strong>Inputs</strong></td>
<td><strong>Outputs</strong></td>
</tr>
<tr>
<td><strong>open</strong></td>
<td>Two Strings 1. File Name 2. “r” for read (for now)</td>
<td>File</td>
</tr>
<tr>
<td><strong>read (On a File)</strong></td>
<td>none</td>
<td>String</td>
</tr>
<tr>
<td><strong>close (On a File)</strong></td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td><strong>split (On a String)</strong></td>
<td>(optional) delimiter</td>
<td>List of Strings</td>
</tr>
</tbody>
</table>

"Inputs" are also called *Arguments.*
Working with Files

1. **Save** `poem.txt` **from the webpage.**
2. **Right-click** and select ‘Properties’
3. **Note the file location** `(C:\Users\Steve\Desktop...)`
4. **In Python,** write an assignment statement that stores the file location as a string.

```python
>>> fileName = "C:\\Users\\Steve\\Desktop\\poem.txt"
```
Working with Files

<table>
<thead>
<tr>
<th>Preloaded Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>type</td>
</tr>
<tr>
<td>open</td>
</tr>
</tbody>
</table>

File is a NEW Type

```python
>>> fileName = "C:\Users\Steve\Desktop\poem.txt"
>>> ```
# Working with Files

## Preloaded Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Expression</td>
<td>Type</td>
</tr>
</tbody>
</table>
| open | Two Strings  
   1. File Name  
   2. “r” for read (for now) | File |

File is a NEW Type

```python
>>> fileName = "C:\Users\Steve\Desktop\poem.txt"
>>> myFile = open(fileName,"r")
```
Working with Files

Preloaded Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Expression</td>
<td>Type</td>
</tr>
</tbody>
</table>
| open     | Two Strings
  1. File Name
  2. “r” for read (for now) | File    |
| read     | none                                        | String  |

(On a File)

```python
>>> fileName = "C:\\Users\\Steve\\Desktop\\poem.txt"
>>> myFile = open(fileName,"r")
```
Working with Files

<table>
<thead>
<tr>
<th>Preloaded Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>type</td>
</tr>
</tbody>
</table>
| open | Two Strings  
1. File Name  
2. “r” for read (for now) | File |
| read (On a File) | none | String |

```python
>>> fileName = "C:\\Users\\Steve\\Desktop\\poem.txt"
>>> myFile = open(fileName,"r")
>>> fileString = myFile.read()
```
Working with Files

Preloaded Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Expression</td>
<td>Type</td>
</tr>
<tr>
<td>open</td>
<td>Two Strings</td>
<td>File</td>
</tr>
<tr>
<td></td>
<td>1. File Name</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2. “r&quot; for read (for now)</td>
<td></td>
</tr>
<tr>
<td>read</td>
<td>none</td>
<td>String</td>
</tr>
<tr>
<td>(On a File)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>close</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>(On a File)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```python
>>> fileName = "C:\\Users\\Steve\\Desktop\\poem.txt"
>>> myFile = open(fileName,"r")
>>> fileString = myFile.read()
>>> myFile.close()
```
Working with Files

```python
>>> shelList = readShel()
>>> shelList
['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

```python
>>> shelList = readShel()
>>> shelList
['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!']
```

**Escape Characters**
- \`\` means interpret the NEXT character differently.
- \n: “new line”
- \’: “apostrophe”
- \t: “tab”
ACT2-2

• Do Task 2
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**
- Compute the average word length of **Moby Dick**
- Find the longest word in **Moby Dick**
Python For Statements (For Loops)

“For each element in list myList, do something”

```python
>>> myList = [1, 2, 3]
```
Python For Statements (For Loops)

“For each element in list myList, do something”

```python
>>> myList = [1, 2, 3]
>>> for element in myList:
...     print element
1
2
3
```
Python \texttt{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”

```python
>>> 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”

```python
>>> myList = [1,2,3]
>>> for num in myList:
...     print num
1
2
3
```
Python *For* Statements (For Loops)

“For each element in list myList, do something”

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

- Do Task 3
def countWordsInShel():
    '''Returns the number of words in the poem.'''
    myList = readShel()
    count = 0
    for word in myList:
        count = count + 1
    print("There are ", count, " words in the poem.")
    return count
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
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
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 |

```python
>>> len(myList)
```
A Shortcut to List Length

Preloaded Functions

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>len</td>
<td>List</td>
</tr>
<tr>
<td></td>
<td>Integer</td>
</tr>
</tbody>
</table>

>>> 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

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

## Python Functions

### Preloaded Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Input Type</th>
<th>Output Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>len</code></td>
<td>List OR String</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>
</tbody>
</table>

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

- Useful in preventing the “int division” confusion: $3/4 \rightarrow 0$, but $\text{float}(3)/\text{float}(4) \rightarrow 0.75$
- **Even** $3/\text{float}(4) \rightarrow 0.75$, **because** if an arithmetic expression involves a float, the result will be a float. $3 + 0.0 \rightarrow 3.0$
- New shorthand: “$\rightarrow$” means “evaluates to”
## Python Functions

<table>
<thead>
<tr>
<th>Preloaded Functions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>len</code></td>
<td>List OR String</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>
<tr>
<td><code>range</code></td>
<td>Two Integers 1. Start Index (Inclusive) 2. End Index (Exclusive)</td>
</tr>
</tbody>
</table>

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

```python
def avgWordLengthInMobyDick():
    '''Gets the average word length in MobyDick.txt'''
    myList = readMobyDick()
    s = 0
    for word in myList:
        s = s + len(word)
    avg = s/float(len(myList))
    return avg
```
Is our Program Correct?

```python
>>> 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

```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

### Numerical Operators

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

### Boolean Operators

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

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

Remember

### Numerical Operators

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

### Boolean Operators

<table>
<thead>
<tr>
<th>Operator</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>
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>
</tr>
</thead>
<tbody>
<tr>
<td>Operator</td>
</tr>
<tr>
<td>Equality</td>
</tr>
<tr>
<td>Inequality</td>
</tr>
<tr>
<td>Less Than</td>
</tr>
<tr>
<td>Less Than or Equal To</td>
</tr>
<tr>
<td>Greater Than</td>
</tr>
<tr>
<td>Greater Than or Equal To</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>
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>Examples</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>and</strong></td>
<td><code>(4&lt;5) and (6&lt;3)</code></td>
<td><code>True and False</code></td>
</tr>
<tr>
<td><strong>or</strong></td>
<td><code>(4&lt;5) or (6&lt;3)</code></td>
<td><code>True or False</code></td>
</tr>
<tr>
<td><strong>not</strong></td>
<td><code>not(4&lt;5)</code></td>
<td><code>not(True)</code></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><strong>and</strong></td>
<td>(4&lt;5) and (6&lt;3)</td>
<td>True and False</td>
<td>False</td>
</tr>
<tr>
<td><strong>or</strong></td>
<td>(4&lt;5) or (6&lt;3)</td>
<td>True or False</td>
<td>True</td>
</tr>
<tr>
<td><strong>not</strong></td>
<td>not(4&lt;5)</td>
<td>not(True)</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></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

### 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>or</td>
<td>(4&lt;5) or (6&lt;3)</td>
<td>True or False</td>
</tr>
<tr>
<td>not</td>
<td>not(4&lt;5)</td>
<td>not(True)</td>
</tr>
</tbody>
</table>

### More Examples

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

## More Examples

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

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

### More Examples

<table>
<thead>
<tr>
<th><strong>More Examples</strong></th>
<th><strong>Result</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>((4&lt;5)) and (((6&lt;3)) or ((5==5)))</td>
<td>\text{True and (False or True)}</td>
</tr>
<tr>
<td>((5==4)) or (\text{not}(6&lt;3))</td>
<td>\text{False or not(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>and</td>
<td>(4&lt;5) and (6&lt;3)</td>
<td>True and False</td>
</tr>
<tr>
<td>or</td>
<td>(4&lt;5) or (6&lt;3)</td>
<td>True or False</td>
</tr>
<tr>
<td>not</td>
<td>not(4&lt;5)</td>
<td>not(True)</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 Statements (\texttt{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)
```
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?

```python
>>> 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