# More Summary Statistics

Mar 3, 2016

## The Big Picture

**Overall Goal**

Build a Concordance of a text

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

**Today: Summary Statistics**

- Review For Loop from the Last Class
- Python/IDLE stuff
- Nitpicky Python details
- A new kind of statement = ‘IF‘
- 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"

```
>>> 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)

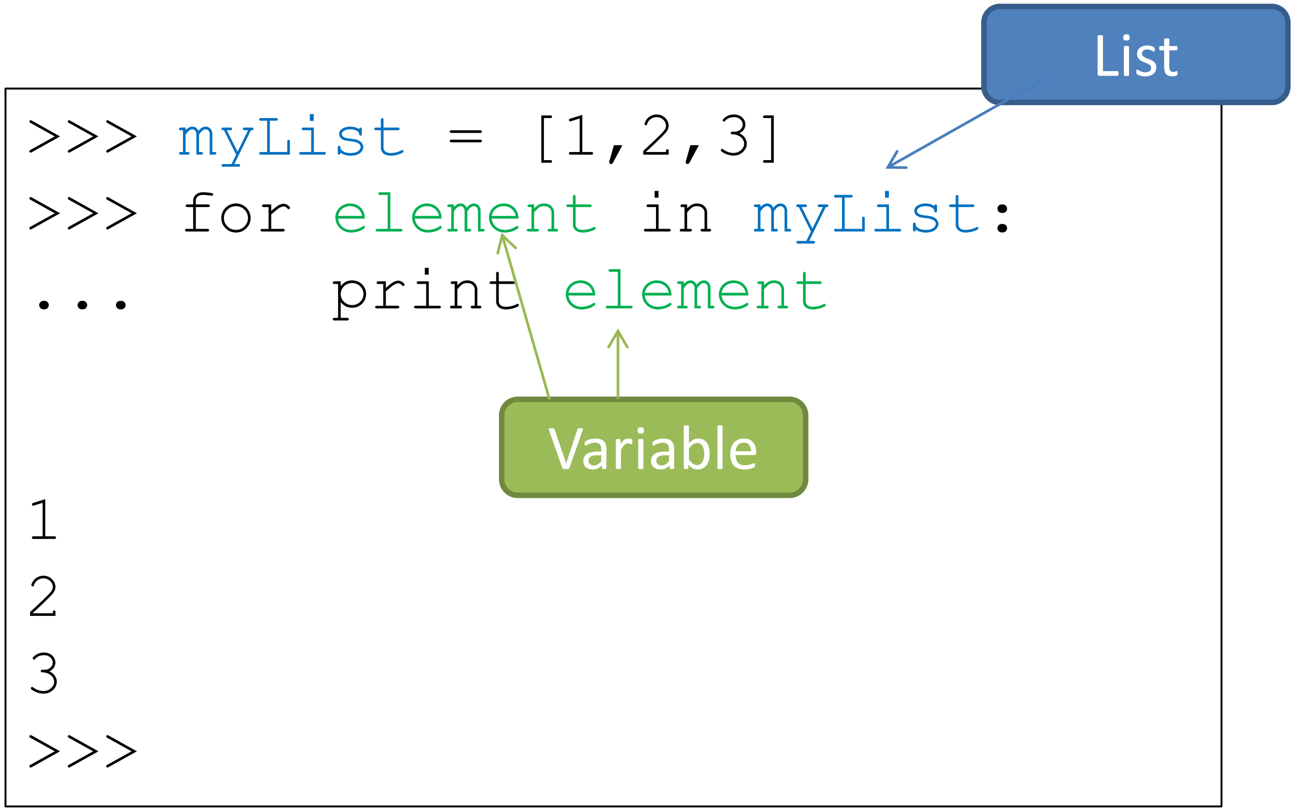
"For each element in list myList, do something"

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



1
2
3
>>>
```

List

# Python `For` Statements (For Loops)

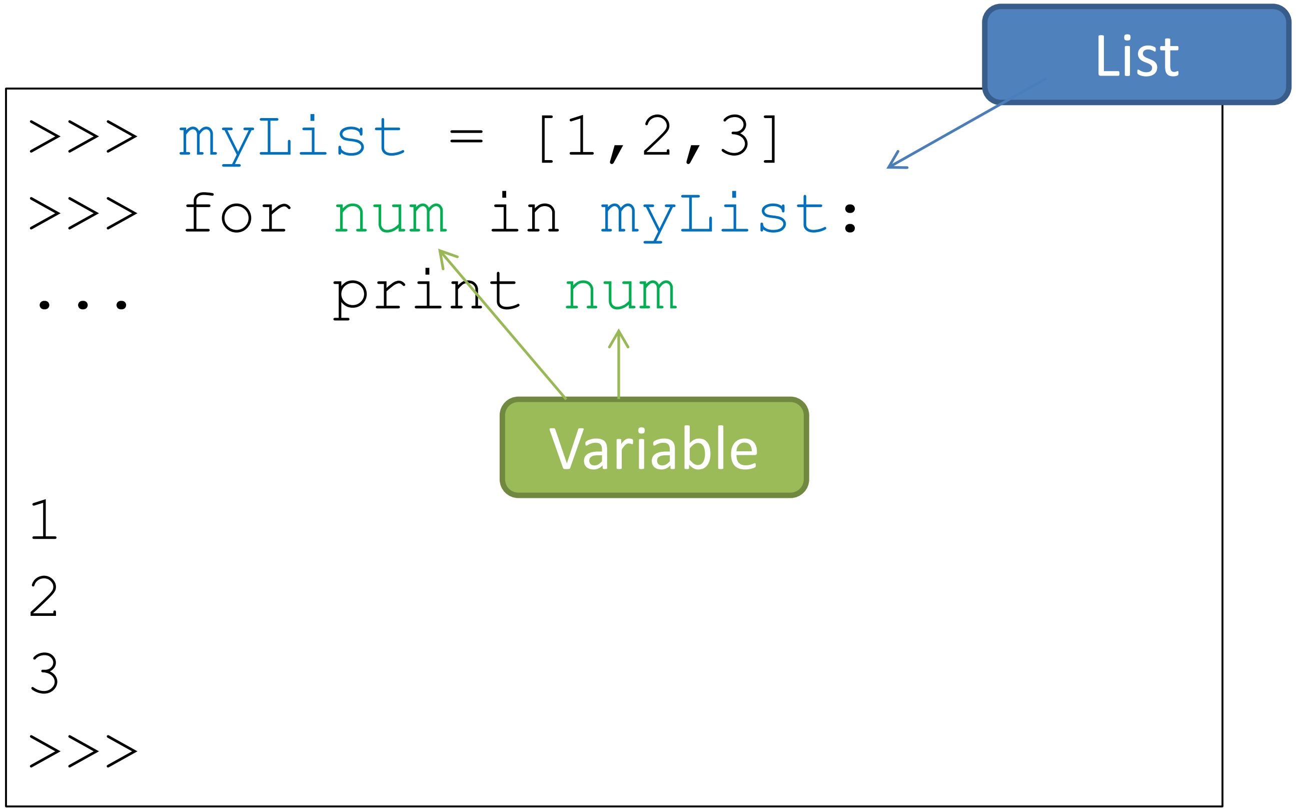
"For each element in list myList, do something"



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


1
2
3
>>>
```

# Python `For` Statements (For Loops)

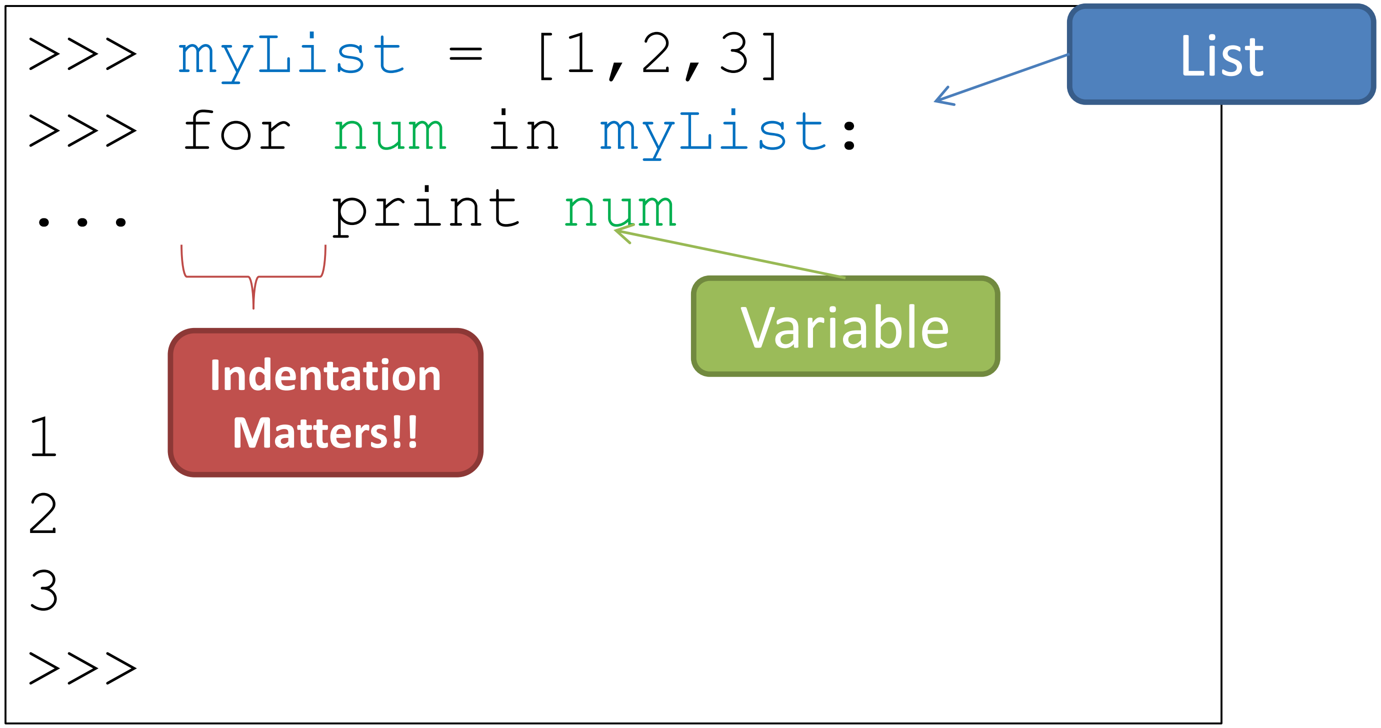
"For each element in list myList, do something"



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


1
2
3
>>>
```

# 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
>>>
```

# 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!**

```python
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

# A Shortcut to List Length

| Preloaded Functions | | |
|---|---|---|
| `len` | List | Integer |

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

# A Shortcut to List Length

| Preloaded Functions | | |
|---|---|---|
| `len` | List | Integer |

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

**From Last Lecture**

- 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 | | |
|---|---|---|
| `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 |

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<br>1. Start Index (Inclusive)<br>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**

- For Loops and Booleans from last time
- Python/IDLE 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 vs. IDLE

- Python is a *language*: a set of rules for what’s “allowable”, much like English grammar (but more sensible).
- This language can be “interpreted”: turned into actions on a computer that do things like read and write files, print output to the screen, etc.
- IDLE is a program that takes input typed in Python and interprets it.

# What IDLE does

- Prints “>>>” and waits for you to type Python
- When you type an expression, IDLE prints out the expression’s value to be helpful
- When you type an assignment, or list-assignment, or function-definition, IDLE just re-prints “>>>”

# IDLE: working directory

- Python has a notion of “current folder” (called “current working directory”)

- If you type

```
>>> f = open (“myfile.txt”, “r”)
```

and `myfile.txt` is in the current directory, the “`open`” will succeed. If not, it’ll fail.

## IDLE: working directory

- Current Directory: place where your Python program is saved!

- General rule: Save in the same folder:
  - Your Python program
  - Your data
- All your "read" statements will work nicely!

# IDLE: working directory

- You can use the full-path of the file

```
>>> f = open (“C:\\Users\\Steve\\...\\myfile.txt”, “r”)
```

i.e., the “full path” to the file

- Or you can use the relative-path of the file

```
>>> f = open (“myfile.txt”, “r”)
```

if the file is in the current working directory

## The Big Picture

**Overall Goal**

Build a Concordance of a text

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

**Today: Summary Statistics**

- Python/IDLE 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

# 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*

# 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

```
>>> 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

Calculates something

- Assignment Statements

*Stores* a value for a variable in memory table

- List-Assignment Sttmts.

*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"

```
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

**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

# 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<br>1. Start Index (Inclusive)<br>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**

- Review For Loop from the Last Class
- Python/IDLE stuff
- Nitpicky Python details
- A new kind of statement
- 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

# Compute the Average Word Length of Moby Dick

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





    return avg
```

# 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/len(myList)
    return avg
```

## Get the Longest Word in <u>Moby Dick</u>

```
def getLongestWordInMobyDick():
    '''Returns the longest word in MobyDick.txt'''





    return longestword
```

## Get the Longest Word in <u>Moby Dick</u>

```
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
```

Is our program correct?

## Next Class

Next time, we'll look at counting the **vocabulary size**, not just the total number of words