Writing your First Python Program

February 28, 2012

Scratch paper will be handy today
Textual Analysis

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
The Big Picture

Overall Goal
Build a Concordance of a text
• Locations of words
• Frequency of words

Today
• Briefly review expressions, assignments, & types
• Learn about defining functions
• Learn how to read in a text file and create a list of words
• Write a program to count the number of words in Moby Dick
1. Expressions
   • Evaluate *input* and returns some *output* (calculator)
2. Assignments: <variable> = <expression>
   • Store the value of the expression in the variable instead of outputting the value.
   • There is *always* an equals sign in an assignment
   • Variables can be named many things
3. Types
   • Integers vs. Floats (Decimals)
   • Strings in single quotes
   • Lists are sets of other types
   • We can index into Strings & Lists
   • Indexed starting at 0!

General Rule: Expressions for a particular type will *output* that same type!
• Do Task 1

1. Expressions
   • Evaluate input and returns some output (calculator)

2. Assignments: <variable> = <expression>
   • Store the value of the expression in the variable instead of outputting the value.
   • There is always an equals sign in an assignment
   • Variables can be named many things

3. Types
   • Integers vs. Floats (Decimals)
   • Strings in single quotes
   • Lists are sets of other types
   • We can index into Strings & Lists
     • Indexed starting at 0!

General Rule:
Expressions for a particular type will output that same type!
The Big Picture

Overall Goal
Build a Concordance of a text
• Locations of words
• Frequency of words

Today
• Briefly review expressions, assignments, & types
• Learn about defining *functions*
• Learn how to read in a text file and create a list of words
• Write a program to count the number of words in *Moby Dick*
Python Functions

Functions are new commands that we define

• Allows us to run many statements at one time.

```python
>>> myList = [2,5,9]
>>> def avg3(someList):
    s = someList[0] + someList[1] + someList[2]
    avg = s/3.0
    return avg

>>> 
```
Python Functions

Functions are new commands that we define

- Allows us to run many statements at one time.

```python
>>> myList = [2, 5, 9]
>>> def avg3(someList):
    s = someList[0] + someList[1] + someList[2]
    avg = s/3.0
    return avg

>>> avg3(myList)
5.333333333333333
>>> myList = [1, 2, 3]
>>> finalValue = avg3(newList)
>>> finalValue
2.0
```
Python Functions

Functions are new commands that we define

• Allows us to run many statements at one time.

```python
>>> myList = [2,5,9]
>>> def avg3(someList):
...     s = someList[0] + someList[1] + someList[2]
...     avg = s/3.0
...     return avg

>>> avg3(myList)
5.333333333333333
>>> myList = [1,2,3]
>>> finalValue = avg3(myList)
>>> finalValue
2.0

WARNING: do not name a variable `sum`. It is a predefined function (it turns purple in IDLE)
```
Python Functions

Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
</table>

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>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

New Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
</table>

```python
>>>
myList = [2, 5, 9]

>>> def avg3(sL):
    s = sL[0] + sL[1] + sL[2]
    avg = s / 3.0
    return avg

>>> avg3(myList)
5.333333333333333

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

>>> finalValue = avg3(myList)

>>> finalValue
2.0
```

“Inputs” are also called Arguments.
Python Functions

Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>myList</td>
<td>[2, 5, 9]</td>
</tr>
</tbody>
</table>

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>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

New Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
</table>

```python
myList = [2, 5, 9]
>>> def avg3(sL):
    s = sL[0] + sL[1] + sL[2]
    avg = s/3.0
    return avg

>>> avg3(myList)
5.333333333333333
>>> myList = [1, 2, 3]
>>> finalValue = avg3(myList)
>>> finalValue
2.0
```
Python Functions

Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>myList</td>
<td>[2, 5, 9]</td>
</tr>
</tbody>
</table>

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>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

New Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>avg3</td>
<td>list</td>
<td>float</td>
</tr>
</tbody>
</table>

>>> myList = [2, 5, 9]
def avg3(sL):
    s = sL[0] + sL[1] + sL[2]
    avg = s/3.0
    return avg

>>> avg3(myList)
5.333333333333333

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

>>> finalValue = avg3(myList)

>>> finalValue
2.0
Python Functions

<table>
<thead>
<tr>
<th>Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>myList</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>New Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>avg3</td>
</tr>
</tbody>
</table>
Python Functions

```python
>>> myList = [2, 5, 9]
>>> def avg3(sL):
...     s = sL[0] + sL[1] + sL[2]
...     avg = s / 3.0
...     return avg

>>> avg3(myList)
5.333333333333333
>>> myList = [1, 2, 3]
>>> finalValue = avg3(myList)
>>> finalValue
2.0
```
Python Functions

Variable Table

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>myList</td>
<td>[2,5,9]</td>
</tr>
<tr>
<td>sL</td>
<td>[2,5,9]</td>
</tr>
<tr>
<td>s</td>
<td>16</td>
</tr>
<tr>
<td>type</td>
<td>expression</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

New Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>avg3</td>
<td>list</td>
<td>float</td>
</tr>
</tbody>
</table>

```python
>>> myList = [2,5,9]
>>> def avg3(sL):
    s = sL[0] + sL[1] + sL[2]
    avg = s/3.0
    return avg

>>> avg3(myList)
5.333333333333333

>>> myList = [1,2,3]
>>> finalValue = avg3(myList)
>>> finalValue
2.0
```
Python Functions

### Variables

<table>
<thead>
<tr>
<th>avg3 Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
</tr>
<tr>
<td>myList</td>
</tr>
<tr>
<td>s</td>
</tr>
<tr>
<td>avg</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>type</th>
<th>expression</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

### New Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>avg3</td>
<td>list</td>
<td>float</td>
</tr>
</tbody>
</table>

```python
>>> myList = [2, 5, 9]
>>> def avg3(sL):
    s = sL[0] + sL[1] + sL[2]
    avg = s/3.0
    return avg

>>> avg3(myList)
5.3333333333
```

```python
>>> myList = [1, 2, 3]
>>> finalValue = avg3(myList)
>>> finalValue
2.0
```
Python Functions

Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>myList</td>
<td>[2, 5, 9]</td>
</tr>
</tbody>
</table>

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>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

New Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>avg3</td>
<td>list</td>
<td>float</td>
</tr>
</tbody>
</table>

```python
>>> myList = [2, 5, 9]
>>> def avg3(sL):
    s = sL[0] + sL[1] + sL[2]
    avg = s/3.0
    return avg

>>> avg3(myList)
5.333333333333333

>>> myList = [1, 2, 3]
>>> finalValue = avg3(myList)

>>> finalValue
2.0
```
Python Functions

>>> myList = [2, 5, 9]
>>> def avg3(sL):
    s = sL[0] + sL[1] + sL[2]
    avg = s/3.0
    return avg

>>> avg3(myList)
5.333333333333333

>>> myList = [1, 2, 3]
>>> finalValue = avg3(myList)
>>> finalValue
2.0

Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>myList</td>
<td>[2, 5, 9]</td>
</tr>
</tbody>
</table>

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>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

New Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>avg3</td>
<td>list</td>
<td>float</td>
</tr>
</tbody>
</table>

Returned value
Python Functions

<table>
<thead>
<tr>
<th>Variables</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>myList</td>
<td>[2, 5, 9]</td>
</tr>
<tr>
<td>myList</td>
<td>[1, 2, 3]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Preloaded Functions</th>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>expression</td>
<td>type</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>New Functions</th>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>avg3</td>
<td>list</td>
<td>float</td>
</tr>
</tbody>
</table>

```python
>>> myList = [2, 5, 9]
>>> def avg3(sL):
...     s = sL[0] + sL[1] + sL[2]
...     avg = s/3.0
...     return avg

>>> avg3(myList)
5.333333333333333
```

```python
>>> myList = [1, 2, 3]
>>> finalValue = avg3(myList)
>>> finalValue
2.0
```
Python Functions

```
>>> myList = [2,5,9]
>>> def avg3(sL):
...     s = sL[0] + sL[1] + sL[2]
...     avg = s/3.0
...     return avg

>>> avg3(myList)
5.333333333333333

>>> myList = [1,2,3]
>>> finalValue = avg3(myList)

>>> finalValue
2.0
```
Python Functions

Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>myList</td>
<td>[2, 5, 9]</td>
</tr>
<tr>
<td></td>
<td>[1, 2, 3]</td>
</tr>
<tr>
<td>finalValue</td>
<td>2.0</td>
</tr>
</tbody>
</table>

New Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>avg3</td>
<td>list</td>
<td>float</td>
</tr>
</tbody>
</table>

```python
>>> myList = [2, 5, 9]
>>> def avg3(sL):
    s = sL[0] + sL[1] + sL[2]
    avg = s/3.0
    return avg

>>> avg3(myList)
5.333333333333333
>>> myList = [1, 2, 3]
>>> finalValue = avg3(myList)
>>> finalValue
2.0
```
Python Functions

### Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>myList</td>
<td>[2, 5, 9]</td>
</tr>
<tr>
<td>myList</td>
<td>[1, 2, 3]</td>
</tr>
<tr>
<td>finalValue</td>
<td>2.0</td>
</tr>
</tbody>
</table>

```python
>>> myList = [2, 5, 9]
>>> def avg3(sL):
    s = sL[0] + sL[1] + sL[2]
    avg = s/3.0
    return avg

>>> avg3(myList)
5.333333333333333
>>> myList = [1, 2, 3]
>>> finalValue = avg3(myList)
>>> finalValue
2.0
```

### New Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>avg3</td>
<td>list</td>
<td>float</td>
</tr>
</tbody>
</table>
Python Functions

```python
>>> def someFunction(inputs):
    output = <some expression>
    return output
```
ACT2-1

• Do Task 2

```python
>>> def someFunction(inputs):
    output = <some expression>
    return output
```
Module Files

Allow us to **save** functions (‘.py’ extension)

- Download `ACT2-1.py` from the website and open it in IDLE. Take a moment to look at it.
- Run...Run Module (or press F5)

- **To write your own file:**
  - File...New Window
  - Write your function definitions. Save the file.
  - Run...Run Module (or press F5)
Polar graph of what stuff happens on which days, based on number of Google results for phrases like “company meeting on <day>.” The relative frequency of <day> in <phrase> is shown by the distance from the center at which <phrase>’s line crosses <day>.

Each curve is normalized to have the same number of total hits—they’re not on the same scale.
The Big Picture

Overall Goal
Build a Concordance of a text
• Locations of words
• Frequency of words

Today
• Briefly review expressions, assignments, & types
• Learn about defining functions
• Learn how to read in a text file and create a list of words
• Write a program to count the number of words in Moby Dick
Working with Files

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

```python
>>> fileName = "C:\Users\Anna\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 Name</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” for read (for now)</td>
<td></td>
</tr>
</tbody>
</table>

File is a NEW Type

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Expression</td>
<td>Type Name</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” for read (for now)</td>
<td></td>
</tr>
</tbody>
</table>
Working with Files

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

>>> fileName = "C:\Users\Anna\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 (On a File) | none | String |

```python
>>> fileName = "C:\Users\Anna\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” 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\Anna\Desktop\poem.txt"
>>> myFile = open(fileName,"r")
>>> fileString = myFile.read()
>>> myFile.close()
```
>> fileString
'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 bananas, rotten peas,
Chunks of sour cottage cheese.
It filled the can, it covered the floor,
It cracked the window and blocked the door
With bacon rinds and chicken bones,
Drippy ends of ice cream cones,
Prune pits, peach pits, orange peel,
Gloppy glumps of cold oatmeal,
Pizza crusts and withered greens,
Soggy beans and tangerines,
Crusts of black burned buttered toast,
...
Because the hour is much too late.
But children, remember Sarah Stout
And always take the garbage out!'
Working with Files

>>> fileString
'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 bananas, rotten peas,
Chunks of sour cottage cheese.
It filled the can, it covered the floor,
Cracked the window and blocked the door
With bacon rinds and chicken bones,
Drippy ends of ice cream cones,
Prune pits, peach pits, orange peel,
Gloppy gloops of cold oatmeal,
Pizza crusts and withered greens,
Soggy beans and tangerines,
Crusts of black burned buttered toast...

Because the hour is much too late.
But children, remember
Sarah Stout
And always take the garbage out!

- Shel Silverstein
## Working with Files

<table>
<thead>
<tr>
<th>Name</th>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>type</strong></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)|                                |                  |
| **close**  | none                           | none             |
| (On a File)|                                |                  |
| **split**  | (optional) delimiter           | List of Strings  |
| (On a String)|                               |                  |

```python
>>> myList = fileString.split()
```
Working with Files

```python
>>> myList
['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!']
```
Activity

• Do Task 3

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

Overall Goal
Build a Concordance of a text
• Locations of words
• Frequency of words

Today
• Briefly review expressions, assignments, & types
• Learn about defining functions
• Learn how to read in a text file and create a list of words
• Write a program to count the number of words in Moby Dick
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 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 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
```
def countWordsInShel():
    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
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
Activity

• Do Task 4
The Big Picture

Overall Goal
Build a Concordance of a text
- Locations of words
- Frequency of words

Today
- Briefly review expressions, assignments, & types
- Learn about defining functions
- Learn how to read in a text file and create a list of words
- Write a program to count the number of words in Moby Dick
- There’s a shortcut...
A Shortcut to List Length

<table>
<thead>
<tr>
<th>Preloaded Functions</th>
<th>“Inputs” are also called Arguments.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Inputs</td>
</tr>
<tr>
<td>type</td>
<td>Expression</td>
</tr>
<tr>
<td>open</td>
<td>Two Strings</td>
</tr>
<tr>
<td></td>
<td>1. File Name</td>
</tr>
<tr>
<td></td>
<td>2. “r” for read (for now)</td>
</tr>
<tr>
<td>read (On a File)</td>
<td>none</td>
</tr>
<tr>
<td>close (On a File)</td>
<td>none</td>
</tr>
<tr>
<td>split (On a String)</td>
<td>(optional) delimiter</td>
</tr>
<tr>
<td>len</td>
<td>List</td>
</tr>
</tbody>
</table>

```python
>>> len(myList)
```
Python `For` Statements (For Loops)

“For each element in list myList, do something”

```python
>>> 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”

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

1
2
3

Preloaded Functions

<table>
<thead>
<tr>
<th>range</th>
<th>Two Integers</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1. Start Index (Inclusive)</td>
</tr>
<tr>
<td></td>
<td>2. End Index (Exclusive)</td>
</tr>
<tr>
<td></td>
<td>List of Integers</td>
</tr>
</tbody>
</table>
Python For Statements (For Loops)

“For each element in list myList, do something”

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

List [0,1,2]

---

Preloaded Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>range</td>
<td>Two Integers</td>
</tr>
<tr>
<td></td>
<td>1. Start Index (Inclusive)</td>
</tr>
<tr>
<td></td>
<td>2. End Index (Exclusive)</td>
</tr>
</tbody>
</table>
Python For Statements (For Loops)

“For each element in list myList, do something”

```python
>>> 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”

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

Q: What if we don’t know the length of the list?
Python for Statements (For Loops)

“For each element in list myList, do something”

```python
>>> 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?
Python **For** Statements (For Loops)

“For each element in list myList, do something”

```python
>>> def printList(list):
    for i in range(0,len(list)):
        print list[i]
    return
```

**Preloaded Functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>range</code></td>
<td>Two Integers</td>
</tr>
<tr>
<td></td>
<td>1. Start Index (Inclusive)</td>
</tr>
<tr>
<td></td>
<td>2. End Index (Exclusive)</td>
</tr>
<tr>
<td></td>
<td>List of Integers</td>
</tr>
</tbody>
</table>
Python For Statements (For Loops)

“For each element in list myList, do something”

```python
>>> def printList(list):
    for i in range(0, len(list)):
        print list[i]
    return

List [0, 1, ..., len(list)-1]

Variable

Indentation Matters!!

Returns NOTHING!
1. Statements
   • Expressions: evaluates *input* and returns some *output*
   • Assignments: `<variable> = <expression>`
   • Print Statements: no parentheses
   • *For* Statements
2. Types
   • Integers & Floats
   • Strings
   • Lists
   • Files
3. Function Definitions
## Function Summary

<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” for read (for now)</td>
<td></td>
</tr>
<tr>
<td>read</td>
<td>none</td>
<td>String</td>
</tr>
<tr>
<td>close</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>split</td>
<td>(optional) delimiter</td>
<td>List of Strings</td>
</tr>
<tr>
<td>len</td>
<td>List</td>
<td>Integer</td>
</tr>
<tr>
<td>range</td>
<td>Two Integers</td>
<td>List of Integers</td>
</tr>
<tr>
<td></td>
<td>1. Start Index (Inclusive)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2. End Index (Exclusive)</td>
<td></td>
</tr>
</tbody>
</table>
General Rules for Writing Functions

• Variables used within function definitions should be one of two things:
  1. An input (also called an argument)
  2. Previously assigned *within* the function def.

• Do not modify arguments within a function definition (define new variables instead)

• Do not have nested function definitions.

• Use only the returned values outside the function definition.