Writing your First Python Program

February 25, 2014

Scratch paper will be handy today
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

ACTACGTCGACTACGATCAC
GATCGCGCGATCACGTATTT
ACGATCAGCTACGATCGATC
TACGATCGTAGCTGTGATCG

CSCI 0931 - Intro. to Comp. for the Humanities and Social Sciences
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 So Far (to be updated/refined!)

1. Expressions
   - Evaluate *input* and returns some *output* (calculator)
2. Variable 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
   - List assignments: `<listvar>[<index>] = <expression>`
3. Types
   - Integers vs. Floats (Decimals)
   - Strings in single quotes
   - Lists are sets of other types
   - We can index into Strings & Lists

General Rule: Expressions for a particular type will *output* that same type!
Subtleties already: names

• We said you could use almost anything as a variable-name
  – Avoid certain words used by Python ("keywords")
    • We’ll mention these as we encounter them
Subtleties: Assignment

• When we enter a formula in cell B2 of a spreadsheet, saying “=A1”, whenever A1 changes, B2 updates
  – That only works for spreadsheets, not Python (nor most other programming languages)

• In Python, assignments “happen once”: the value of the right hand side, right now, is assigned to the left hand side
Subtleties: Assignment (2)

• Details. In the assignment statement

\[ x = y + 3 \]

– The expression on the right is evaluated;
  • if there are variable names there, the values are looked up in the memory table
  – If there’s not already an assigned value in the memory table for the variable on the left (\( x \) in this case), Python makes space for it
  – The computed value is placed in the memory as the value for the variable
Pictorial version of assignment

```python
>>> x = 5
```

- Evaluate RHS: 5
- There’s no memory spot for `x`: create one

<table>
<thead>
<tr>
<th>Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>-------</td>
</tr>
<tr>
<td><code>a</code></td>
</tr>
</tbody>
</table>

- Put the value of the RHS in the “value” table

<table>
<thead>
<tr>
<th>Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>-------</td>
</tr>
<tr>
<td><code>a</code></td>
</tr>
<tr>
<td><code>x</code></td>
</tr>
</tbody>
</table>
Pictorial version of assignment, v2

```python
>>> x = 5

• Evaluate RHS: 5
• There’s already a memory spot for `x`: do nothing

```
\[ x = x + 1, \text{ pictorially, } v1 \ (x \text{ already defined}) \]

>>> 

- Evaluate RHS: \( x+1 \)
  - Lookup \( x \), get 0
  - Add 1, to get 1
- There’s already a memory spot for \( x \): do nothing

### Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x )</td>
<td>int</td>
<td>0</td>
</tr>
</tbody>
</table>

- Put the value of the RHS in the “value” table

### Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x )</td>
<td>int</td>
<td>1</td>
</tr>
</tbody>
</table>
x = x+1, pictorially, v2 (x not defd)

- Evaluate RHS: x+1
  - Lookup x...it’s not there
  - ERROR!

<table>
<thead>
<tr>
<th>Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>-----------</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
What’s “evaluate” mean?

- Lookup variable names to find values
- Do math or string or list operations in the order specified to combine these values and get a result
Key Points

• Some variable names should not be used.
• Variables don’t have values until you assign them (unlike spreadsheets, where you can refer to cell A10 before putting anything in there)
• Assignment is a 3-step process
  – Evaluate right hand side
  – Make room in memory table if needed
  – Place value from first step in table
List indexing

• When you have a list:
  >>> myList = [1, 3, 5, 4, 6]

• You can refer to individual items:
  >>> myList[0]
  1

• Or pieces of it
  >>> myList[0:3]
  [1, 3, 5]
List Indexing, 2

From previous slide:

```python
>>> myList = [1, 3, 5, 4, 6]
```

- Pieces of list:
  - Can also use `[:3]` or `2:` to refer to “stuff up to but not including item 3” or “stuff including and after item 2”

```python
>>> myList[:3]
[1, 3, 5]
>>> myList[2:]
[5, 4, 6]
```

- You’ll want this for HW this weekend
Those “pieces of lists” (sometimes called “slices”) can appear on the left-hand side of an assignment.

```python
>>> myList = [1, 3, 5, 4, 6]
>>> myList[0] = 5
>>> myList
[5, 3, 5, 4, 6]
>>> myList[0:2] = [9]
>>> myList
[9, 5, 4, 6]
>>> myList[0:3] = []
>>> myList
[6]
```
ACT2-1

• Do Task 1
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 multi-step operations that we define

• Allows us to execute many statements in sequence.

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

>>> mylist = [2,5,9]
```
Python Functions

• Functions are multi-step operations that we define

• Allows us to execute many statements in sequence.

```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
```
Python Functions

• Functions are multi-step operations that **we** define

• Allows us to execute many statements in sequence.

```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

Define a new function using the keyword `def`:

- Can take zero or more inputs (called *arguments*)
- Does some computation using the inputs
- Returns a value
- Form of a function definition:

  ```python
  def <functionName>(arg1, ..., argn) :
      statement1
      statement2
      ...
      statementn
      return <value>
  ```

- Arg1, ..., argn must be variable names
- There might be none of them...but parentheses are still required
Expanded model of Python execution

• There’s not just one “table” in memory
• There’s a table for variable names and their values
• There’s a table for function names and their associated functions
  – That table comes in two part: preloaded (i.e., part of Python) and user-defined (which we call “new functions”)

CSCI 0931 - Intro. to Comp. for the Humanities and Social Sciences
Expanded model, 2

• When a function is *used* (or “called” or “invoked” or ...), a further *temporary* memory table is created

• This table disappears when the function “returns” (or terminates, or finishes)

• Why? Because doing it this way prevents a TON of programming mistakes!

• More details later: let’s see it in action
## Python Functions

### Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></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></td>
<td></td>
<td></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>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>myList</td>
<td>list</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>
</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>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>myList</td>
<td>list</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

"Inputs" are also called Arguments.
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
```

### 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>

### 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

<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>Python Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;&gt;&gt; myList = [2, 5, 9]</td>
</tr>
</tbody>
</table>
| >>> 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               |

<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

### Variables

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

### Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>sL</td>
<td>list</td>
<td>[2, 5, 9]</td>
</tr>
<tr>
<td>s</td>
<td>int</td>
<td>16</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>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>sL</td>
<td>list</td>
<td>[2,5,9]</td>
</tr>
<tr>
<td>s</td>
<td>int</td>
<td>16</td>
</tr>
<tr>
<td>avg</td>
<td>float</td>
<td>5.33333</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)
5.333333333333333

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

Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>myList</td>
<td>list</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>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>myList</td>
<td>list</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>
</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 Functions

### Variables

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

```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
```

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

### Variables

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

Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>myList</td>
<td>list</td>
<td>[1,2,3]</td>
</tr>
<tr>
<td>finalValue</td>
<td>float</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

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>

Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>myList</td>
<td>list</td>
<td>[1, 2, 3]</td>
</tr>
<tr>
<td>finalValue</td>
<td>float</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
```
Python Functions

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

- **Function Definition**
- **Function Inputs / Arguments** (Optional)
- **Function Output** (Optional)
- **Indentation Matters!!**
ACT2-1

• Do Task 2

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

Allow us to **save** code (‘.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)
Break
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
### Activity

- **Do Task 3**

<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>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>read</td>
</tr>
<tr>
<td>(On a File)</td>
</tr>
<tr>
<td>close</td>
</tr>
<tr>
<td>(On a File)</td>
</tr>
<tr>
<td>split</td>
</tr>
<tr>
<td>(On a String)</td>
</tr>
</tbody>
</table>
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

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>

>>> fileName = "C:\\Users\\Steve\\Desktop\\poem.txt"

File is a NEW Type
## 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>
</tbody>
</table>
| open | Two Strings  
1. File Name  
2. “r” for read (for now) | File |

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

File is a NEW Type
## 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 (On a File)</td>
<td>none</td>
<td>String</td>
</tr>
</tbody>
</table>

```python
>>> fileName = "C:\\Users\\Steve\\Desktop\\poem.txt"
>>> myFile = open(fileName,"r")
```
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</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 (On a File)</td>
<td>none</td>
<td>String</td>
</tr>
</tbody>
</table>

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

>>> fileString
'Sarah Cynthia Sylvia Stout
Would not take the garbage out!
'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\nWould not take the garbage out!\nShe\'d scour the pots and scrape the pans,\nCandy the yams and spice the hams,\nAnd though her daddy would scream and shout,\nShe simply would not take the garbage out.\nAnd so it piled up to the ceilings:
Coffee grounds, potato peelings, brown bananas, rotten peas, chunks of sour cottage cheese.\nIt 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\nAnd always take the garbage out!' - Shel Silverstein

“Escape” Characters
• `\` means interpret the NEXT character differently.
• `\n`: “new line”
• `\’`: “apostrophe”
• `\t`: “tab”
• `\\`: a single backslash (\)
• Many others you probably won’t use
### Working with Files

<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>

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

>>> wordList
['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!']
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
• Next time: Write a program to count the number of words in Moby Dick
More Python practice due Tuesday

• But not a lot (since project is due Monday)
If we have time...
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
>>> 
```

List
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():

    return count
Activity

• Do Task 4
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
    return count
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

### Preloaded Functions

<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>
<tr>
<td><strong>open</strong></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><strong>read</strong></td>
<td>none</td>
<td>String</td>
</tr>
<tr>
<td>(On a File)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>close</strong></td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>(On a File)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>split</strong></td>
<td>(optional) delimiter</td>
<td>List of Strings</td>
</tr>
<tr>
<td>(On a String)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>print</strong></td>
<td>Expression</td>
<td>none</td>
</tr>
<tr>
<td><strong>len</strong></td>
<td>List</td>
<td>Integer</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
```

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

Preloaded Functions

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

List [0, 1, 2]
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>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
>>> def printList(list):
    for i in range(0, len(list)):
        print(list[i])
    return

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

Indentation Matters!!

Variable

Returns NOTHING!
Python Summary

1. Statements
   • Expressions: evaluates *input* and returns some *output*
   • Assignments: `<variable> = <expression>`
   • Print Statements
   • For Statements
   • List-assignment (more later)
   • Function definitions

2. Types
   • Integers & Floats
   • Strings
   • Lists
   • Files
# 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 1. File Name 2. “r” for read (for now)</td>
<td>File</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>print</td>
<td>Expression</td>
<td>none</td>
</tr>
<tr>
<td>range</td>
<td>Two Integers 1. Start Index (Inclusive) 2. End Index (Exclusive)</td>
<td>List of Integers</td>
</tr>
</tbody>
</table>

“Inputs” are also called *Arguments.*
General Rules for Writing Functions in CSCS0931

• These rules are here to help you!

• 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.*