Writing your First Python Program

Feb 25, 2016

Today's Class

- Brief discussion on Project 1
- More Python!

Get **Full** Credit on Project 1

- Use sheets for intermediate results, interactivity
- Display your data in different ways (graph, table, etc.)
- Follow the rules:
  - only data, parameters, labels, and formulas.
- Use “notes” on cells for explanation
  - only if anything out of the ordinary
- Try to avoid “hand work” and use formulas instead!

Get **Full** Credit on Project 1

- Remember to address your claim
  - My hypothesis is correct/incorrect because …
  - X% of the time, my hypothesis was correct…
- Be sure to explain what obstacles you encountered
- Have a "Discussion and Conclusion" (2pts)
  - Reflect on things like:
    - "Is this data too unreliable for me to trust the conclusion?"
    - "Was a threshold of 80% really reasonable?"
    - "Did eliminating countries that lacked data for any single year make the analysis compromised somehow?"
- Look at the rubric!

Intermediate Results

Put your **raw data on its own sheet** and **refer to it using a formula** when you do your analysis on other sheets.

Intermediate Results

Use a **new sheet** when the current sheet already has a table with some meaningful data in it. (10pts)

Don’t lose that data.

Interactivity

Use **data validation**, probably with a list (pull down), to add some interactive component (5pts).

See ACT1-3 for an example using MATCH and OFFSET.

Presentation

Try out different ways to present your data.

Make a chart from your final results.

If you're not sure what to do, ask a TA or me.

Address Your Hypothesis

Remember to relate your results back to the hypothesis.

Was it true?

Was it true in some cases?

Why might it have been false?

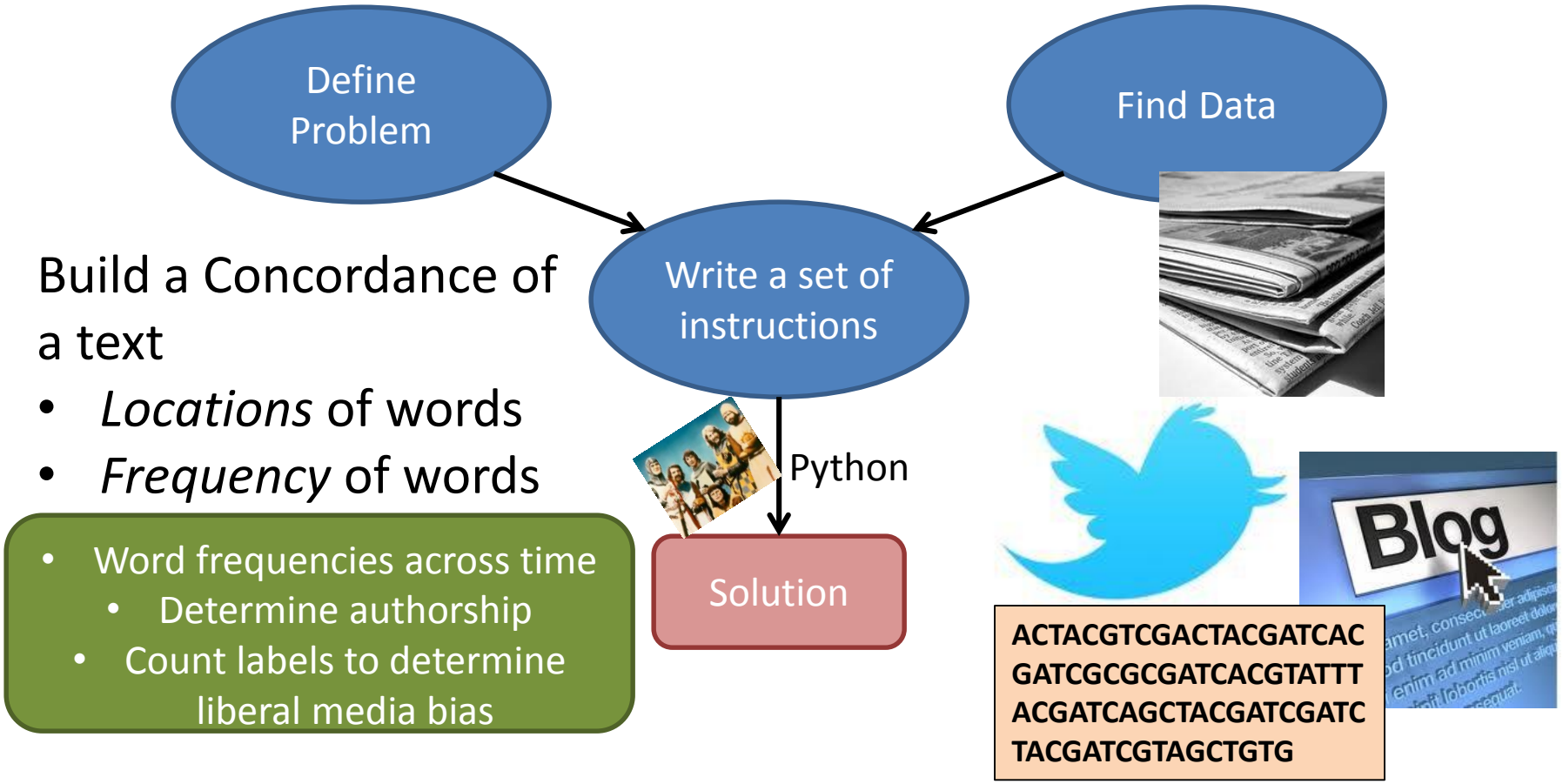
Obstacles and Reflection

What was difficult? What are the limitations of the analysis you did?

You **must** reflect on your project and what you learned.

Check the rubric **before you submit**

Textual Analysis



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

Last Class

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

Expressions for a particular type will *output* that same type! Floats have higher priority

Interactive sessions

```
>>> myList = [2,5,9]
>>> mySum = myList[0] + myList[1] + myList[2]
>>> mySum
16
>>> myAvg = mySum/3
>>> myAvg
5.33333333
>>>
>>> x = 3
>>> y = x +  1
>>> y
4
>>> x = 4
>>> y
4
```

Non-interactive version

Example.py

```
# This program calculates the average of numbers in
# the 3-element list myList below,
# does semi-uselesss things with variable x,
# and prints "4" twice

myList = [2,5,9]
mySum = myList[0] + myList[1] + myList[2]
myAvg = mySum/3

print(myAvg)
x = 3
y = x + 1
print(y)
x = 4
print(y)
```

Module Files

Allow us to **save** code ('.py' extension)

- Download `Example.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`)

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

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

Variables		
Name	Type	Value
a	int	3

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

Variables		
Name	Type	Value
a	int	3
x	int	

- Put the value of the RHS in the "value" table

Variables		
Name	Type	Value
a	int	3
x	int	5

Pictorial version of assignment, v2

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

Variables		
Name	Type	Value
`x`	`int`	`0`

- Evaluate RHS: 5
- There's already a memory spot for `x`: do nothing

Variables		
Name	Type	Value
`x`	int	0

- Put the value of the RHS in the "value" table

Variables		
Name	Type	Value
`x`	int	5

x = x+1 (x already defined)

```
>>>
```

Variables		
Name	Type	Value
`x`	`int`	`0`

- Evaluate RHS: x+1
  - Lookup x, get 0
  - Add 1, to get 1
- There's already a memory spot for `x`: do nothing

Variables		
Name	Type	Value
`x`	int	0

- Put the value of the RHS in the "value" table

Variables		
Name	Type	Value
`x`	int	1

x = x+1 (x not previously defined)

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

Variables		
Name	Type	Value

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

- 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

- To get a **range** of elements from a list, use the expression `>>> myList[i:j]` where `i` is the start index (inclusive) and `j` is the end index (**exclusive**).

```
>>> myList
[5, 4, 15]
>>> myList[0:2]
[5, 4]
>>> myList[1:3]
[4, 15]
>>> newList = [2,5,29,1,9,59,3]
>>> newList
[2, 5, 29, 1, 9, 59, 3]
>>> newList[2:6]
[29, 1, 9, 59]
```

List Indexing

- **Indexing** and **ranges** also work on Strings.

```
>>> myString = “hi there”
>>> myString
'hi there'
>>> myString[0]
'h'
>>> myString[5]
'e'
>>> myString[6]
'r'
>>> myString[0:6]
'hi the'
```

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, reloaded

From previous slide:

```
>>> 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"

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

- Handy for the cookie monster task on HW

List Indexing, more tricks

- Those “pieces of lists” (sometimes called “slices”) can appear on the left-hand side of an assignment

```
>>> 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]
```

Note the braces! Needed for slice assignment.

ACT2-1

- Do Task 1

The Big Picture

Overall Goal

Build a Concordance of a text

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

Steps

- 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.

```
>>> myList = [2,5,9]
>>> def avg3(someList):
        s = someList[0] + someList[1] + someList[2]
        avg = s/3
        return avg

>>>
```

Python Functions

- Functions are multi-step operations that **we** define
- Allows us to execute many statements in sequence.

```
>>> myList = [2,5,9]
>>> def avg3(someList):
        s = someList[0] + someList[1] + someList[2]
        avg = s/3
        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.

```
>>> myList = [2,5,9]
>>> def avg3(someList):
        s = someList[0] + someList[1] + someList[2]
        avg = s/3
        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:

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

- **arg1**, ..., **arg*n*** must be variable names
- There might be none of them...but parentheses are still required

Expanded model of Python execution

- There's a table for *variable* names and their values

- There's a table for *function* names and their associated functions
  - That table has two parts:
    - preloaded (i.e., part of Python)
    - user-defined (which we call "new functions")

Expanded model of Python execution

- 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		
Name	Type	Value

Preloaded Functions		
Name	Inputs	Outputs
`type`	expression	type
…		

New Functions		
Name	Inputs	Outputs

```
>>> myList = [2,5,9]
>>> def avg3(sL):
        s = sL[0] + sL[1] + sL[2]
        avg = s/3
        return avg

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

Python Functions

Variables		
Name	Type	Value
myList	list	[2,5,9]

Preloaded Functions		
Name	Inputs	Outputs
type	expression	type
…		

New Functions		
Name	Inputs	Outputs

Assignment statement

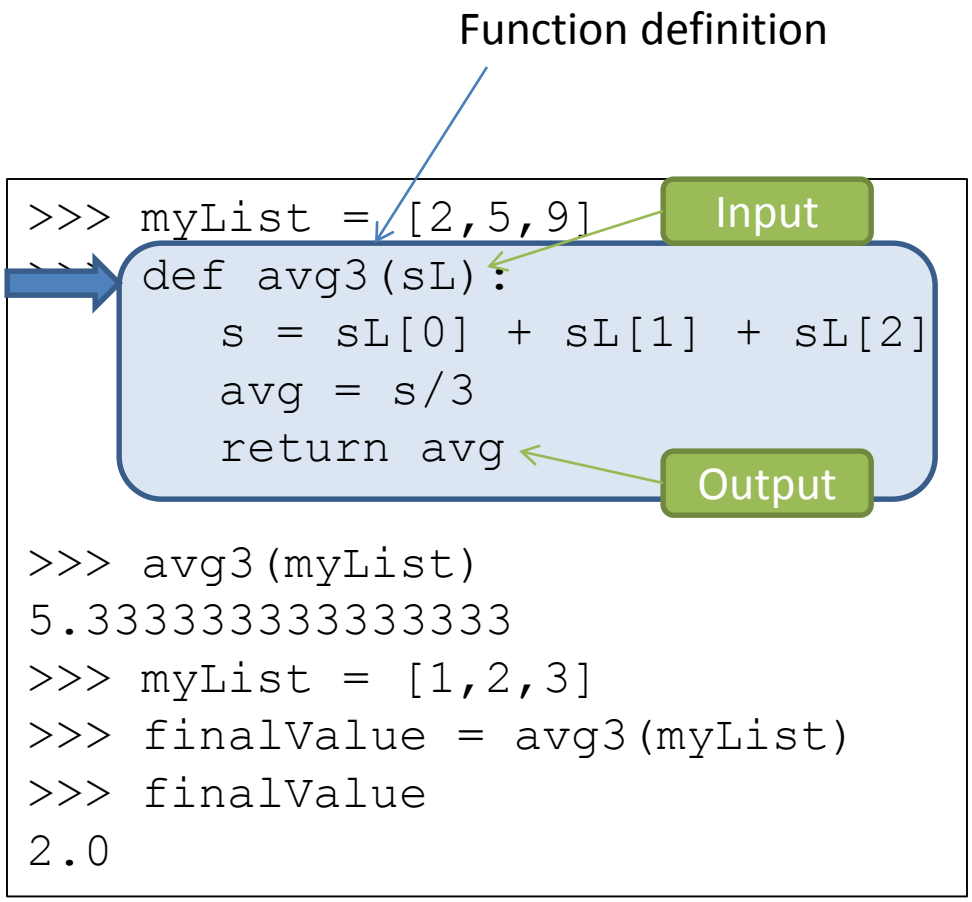
```
>>> myList = [2,5,9]
>>> def avg3(sL):
        s = sL[0] + sL[1] + sL[2]
        avg = s/3
        return avg

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

Python Functions

Variables		
Name	Type	Value
`myList`	`list`	`[2,5,9]`

Preloaded Functions		
Name	Inputs	Outputs
`type`	expression	type
…		

New Functions		
Name	Inputs	Outputs
`avg3`	list	float



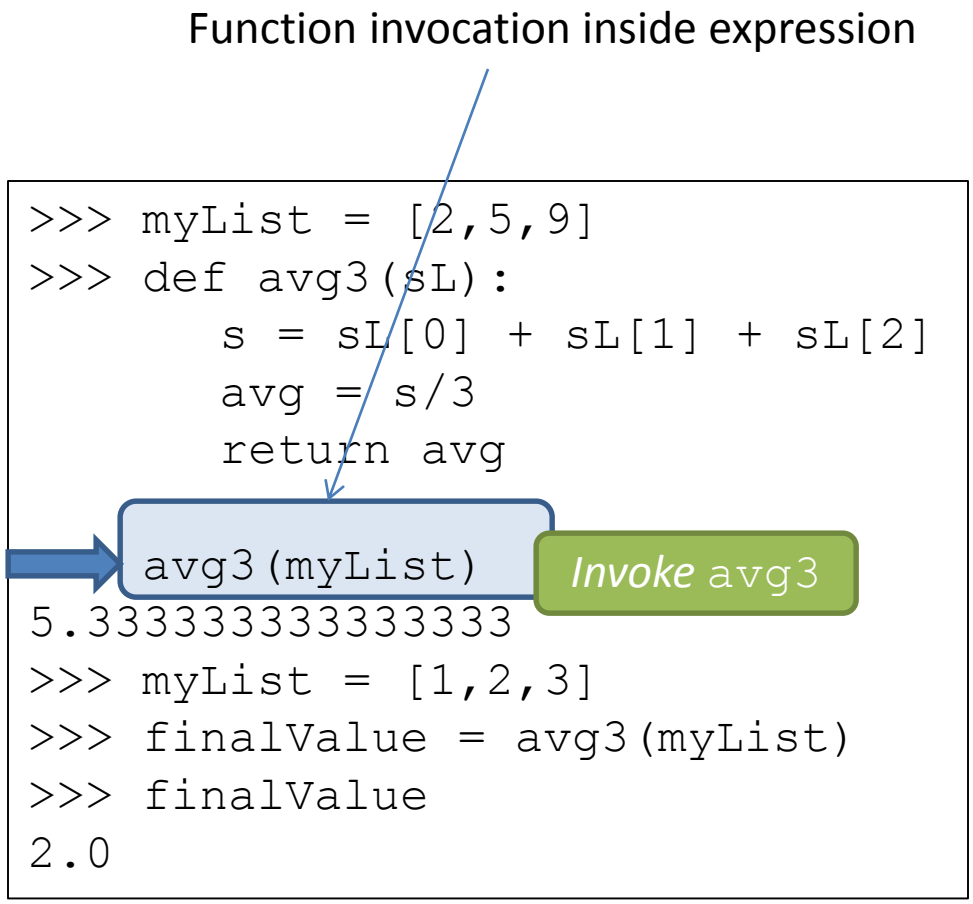
"Inputs" are also called *Arguments.*

Python Functions

Variables		
Name	Type	Value
`myList`	`list`	`[2,5,9]`

Preloaded Functions		
Name	Inputs	Outputs
`type`	expression	type
…		

New Functions		
Name	Inputs	Outputs
`avg3`	list	float



Python Functions

avg3 Variables

Name	Type	Value
sL	list	[2,5,9]

type	expression	type
…		

New Functions

Name	Inputs	Outputs
avg3	list	float

```
>>> myList = [2,5,9]
>>> def avg3(sL):
        s = sL[0] + sL[1] + sL[2]
        avg = s/3
        return avg

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

Invoke `avg3`

Python Functions

avg3 Variables

Name	Type	Value
sL	list	[2,5,9]
s	int	16

type	expression	type
…		

New Functions

Name	Inputs	Outputs
avg3	list	float

```
>>> myList = [2,5,9]
>>> def avg3(sL):
        s = sL[0] + sL[1] + sL[2]
        avg = s/3
        return avg

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

Python Functions

avg3 Variables

Name	Type	Value
sL	list	[2,5,9]
s	int	16
avg	float	5.33333

type	expression	type
…		

New Functions

Name	Inputs	Outputs
avg3	list	float

```
>>> myList = [2,5,9]
>>> def avg3(sL):
        s = sL[0] + sL[1] + sL[2]
        avg = s/3
        return avg

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

Python Functions

Variables		
Name	Type	Value
`myList`	`list`	`[2,5,9]`

Preloaded Functions		
Name	Inputs	Outputs
`type`	expression	type
…		

New Functions		
Name	Inputs	Outputs
`avg3`	list	float

```
>>> myList = [2,5,9]
>>> def avg3(sL):
        s = sL[0] + sL[1] + sL[2]
        avg = s/3
    return avg

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

Python Functions

Variables		
Name	Type	Value
`myList`	`list`	`[2,5,9]`

Preloaded Functions		
Name	Inputs	Outputs
`type`	expression	type
…		

New Functions		
Name	Inputs	Outputs
`avg3`	list	float

```
>>> myList = [2,5,9]
>>> def avg3(sL):
        s = sL[0] + sL[1] + sL[2]
        avg = s/3
        return avg

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

Returned value

Python Functions

Variables		
Name	Type	Value
myList	list	~~[2,5,9]~~ [1,2,3]

Preloaded Functions		
Name	Inputs	Outputs
type	expression	type
…		

New Functions		
Name	Inputs	Outputs
avg3	list	float

```
>>> myList = [2,5,9]
>>> def avg3(sL):
        s = sL[0] + sL[1] + sL[2]
        avg = s/3
        return avg

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

Python Functions

Variables

avg3 Variables

Name	Type	Value
sL	list	[1,2,3]

type	expression	type
…		

New Functions

Name	Inputs	Outputs
avg3	list	float

```
>>> myList = [2,5,9]
>>> def avg3(sL):
        s = sL[0] + sL[1] + sL[2]
        avg = s/3
        return avg

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

Invoke `avg3`

Python Functions

Variables

avg3 Variables

Name	Type	Value
sL	list	[1,2,3]
s	int	6

type	expression	type
…		

New Functions

Name	Inputs	Outputs
avg3	list	float

```
>>> myList = [2,5,9]
>>> def avg3(sL):
        s = sL[0] + sL[1] + sL[2]
        avg = s/3
        return avg

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

Python Functions

Variables

avg3 Variables

Name	Type	Value
sL	list	[1,2,3]
s	int	6
avg	float	2.0

type	expression	type
…		

New Functions

Name	Inputs	Outputs
avg3	list	float

```
>>> myList = [2,5,9]
>>> def avg3(sL):
        s = sL[0] + sL[1] + sL[2]
        avg = s/3
        return avg

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

Python Functions

Variables

avg3 Variables

Name	Type	Value
sL	list	[1,2,3]
s	int	6
avg	float	2.0

type	expression	type
…		

New Functions

Name	Inputs	Outputs
avg3	list	float

```
>>> myList = [2,5,9]
>>> def avg3(sL):
        s = sL[0] + sL[1] + sL[2]
        avg = s/3
    return avg

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

Python Functions

Variables		
Name	Type	Value
`myList`	`list`	`[1,2,3]`
`finalValue`	`float`	`2.0`

New Functions		
Name	Inputs	Outputs
`avg3`	list	float

```
>>> myList = [2,5,9]
>>> def avg3(sL):
        s = sL[0] + sL[1] + sL[2]
        avg = s/3
        return avg

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

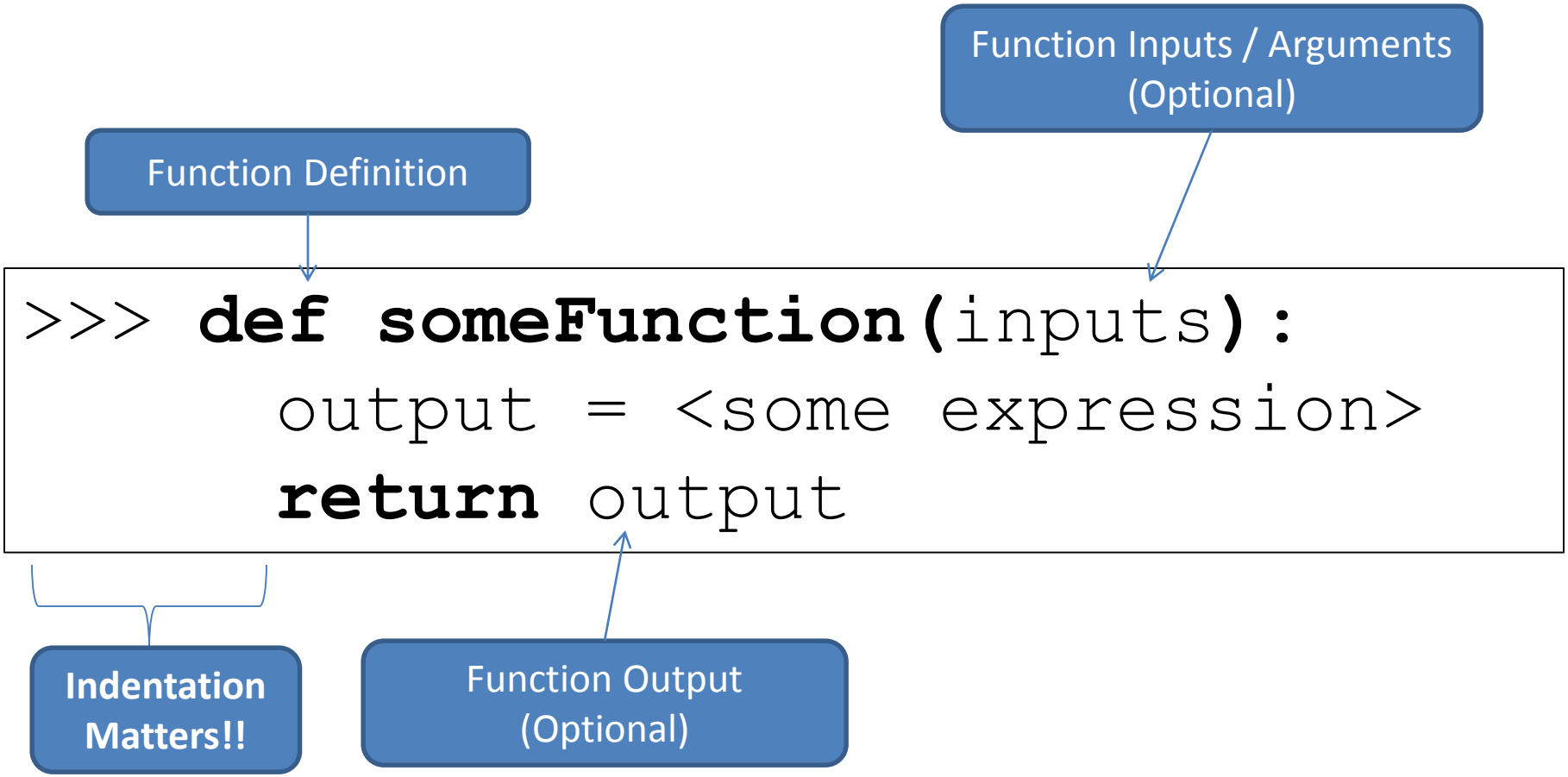
Python Functions

Variables		
Name	Type	Value
`myList`	`list`	`[1,2,3]`
`finalValue`	`float`	`2.0`

New Functions		
Name	Inputs	Outputs
`avg3`	list	float

```
>>> myList = [2,5,9]
>>> def avg3(sL):
        s = sL[0] + sL[1] + sL[2]
        avg = s/3
        return avg

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

Python Functions



ACT2-1

- Do Task 2

```
>>> 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`)

The Big Picture

Overall Goal

Build a Concordance of a text

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

Steps

- 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

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.*