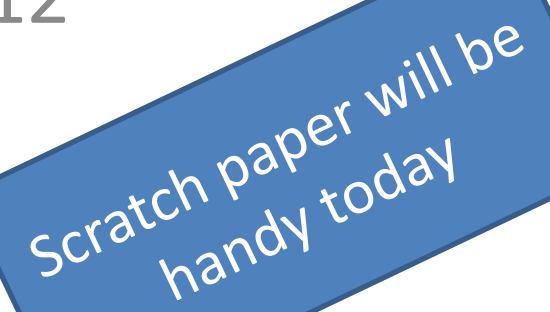


Writing your First Python Program

February 28, 2012



Scratch paper will be handy today

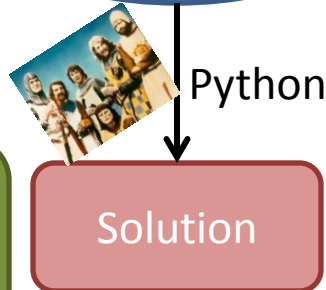
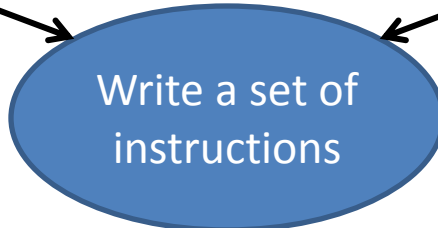
Textual Analysis



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



Solution



```
ACTACGTCGACTACGATCAC
GATCGCGCGATCACGTATTT
ACGATCAGCTACGATCGATC
TACGATCGTAGCTGTGATCG
```

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

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!

ACT2-1

- 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.

```
>>> 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.

```
>>> 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.

```
>>> 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

Name	Value

Preloaded Functions

Name	Inputs	Outputs
<code>type()</code>	expression	type
...		

New Functions

Name	Inputs	Outputs



```
>>> 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

Name	Value
myList	[2, 5, 9]

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/3.0
    return avg

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

Python Functions

Variables

Name	Value
myList	[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/3.0
    return avg
>>> avg3(myList)
5.333333333333333
>>> myList = [1, 2, 3]
>>> finalValue = avg3(myList)
>>> finalValue
>>> 2.0
```

Input

Output

Python Functions

Variables

Name	Value
myList	[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/3.0
    return avg
```



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

Invoke avg3

Python Functions

avg3 Variables		
Name	Value	
sL	[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/3.0
    return avg
>>> avg3(myList)
5.333333333333333
>>> myList = [1, 2, 3]
>>> finalValue = avg3(myList)
>>> finalValue
>>> 2.0
```

Invoke avg3

Python Functions

avg3 Variables		
Name	Value	
sL	[2, 5, 9]	
s	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/3.0
    return avg

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

Invoke avg3

Python Functions

avg3 Variables		
Name	Value	
sL	[2, 5, 9]	
s	16	
avg	5.333333333333	

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/3.0
    return avg
>>> avg3(myList)
5.333333333333333
>>> myList = [1, 2, 3]
>>> finalValue = avg3(myList)
>>> finalValue
>>> 2.0
```

Invoke avg3

Python Functions

Variables

Name	Value
myList	[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/3.0
    return avg
```



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

Invoke avg3

Python Functions

Variables

Name	Value
myList	[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/3.0
    return avg
```

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

Returned value

Python Functions

Variables

Name	Value
myList	[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/3.0
        return avg

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

Python Functions

Variables

avg3 Variables

Name	Value
sL	[1, 2, 3]
s	6
avg	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/3.0
    return avg

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

Invoke avg3

Python Functions

Variables

Name	Value
myList	[2, 5, 9] [1, 2, 3]
finalValue	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/3.0
    return avg

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

Python Functions

Variables

Name	Value
myList	[2, 5, 9] [1, 2, 3]
finalValue	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/3.0
    return avg

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

Python Functions

Function Definition

Function Inputs
(Optional)

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

Indentation
Matters!!

Function Output
(Optional)

ACT2-1

- Do Task 2

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

Module Files

Allow us to **save** functions (`' .py'` extension)

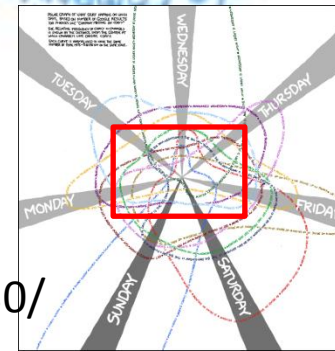
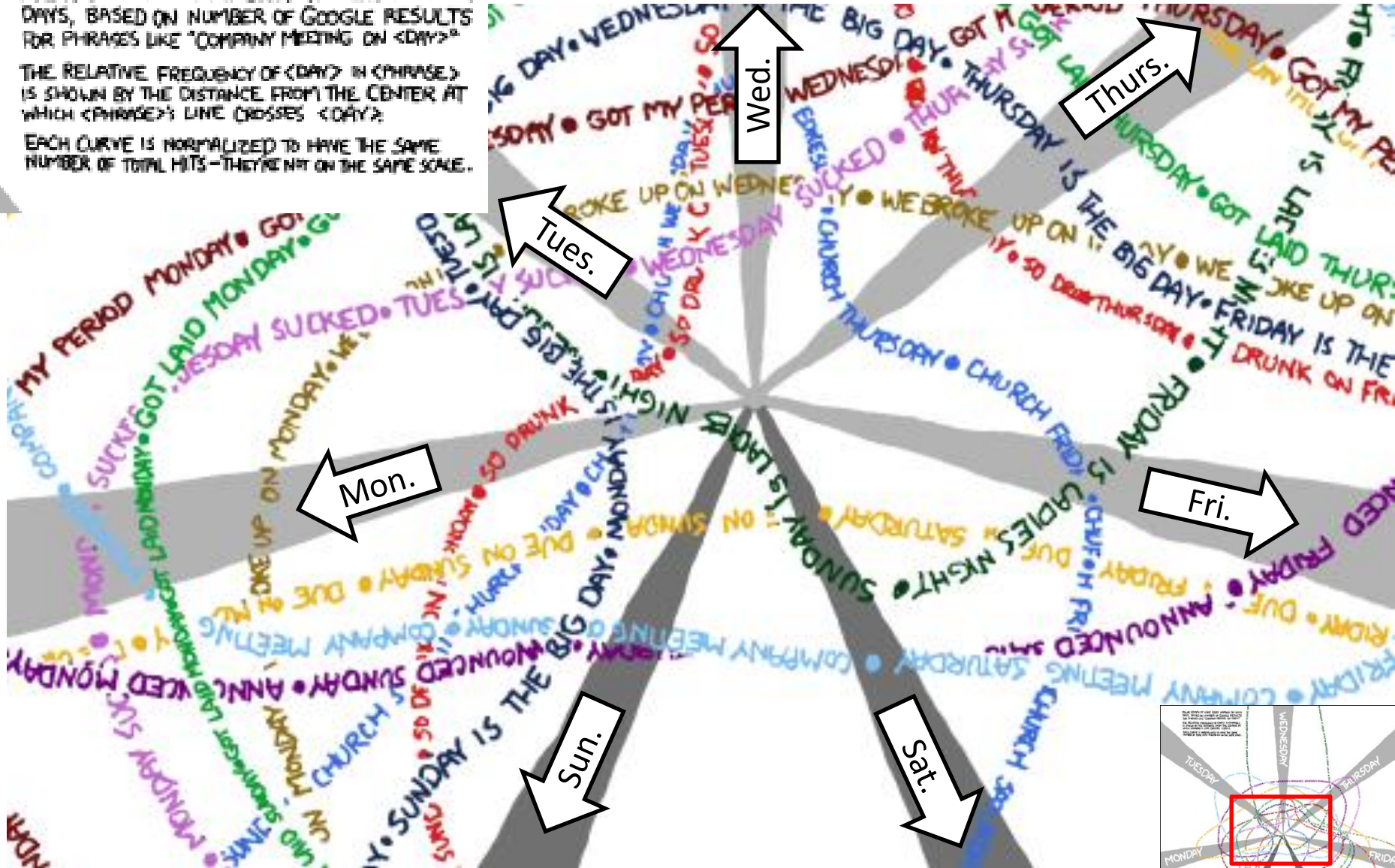
- Download `ACT2-1.py` from the website and open it in IDLE. Take a moment to look at it.
- `Run...Run Module` (or press `F5`)
- **To write your own file:**
 - `File...New Window`
 - Write your function definitions. Save the file.
 - `Run...Run Module` (or press `F5`)

Break

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.

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

Working with Files

Preloaded Functions

Name	Inputs	Outputs
type	Expression	Type Name
open	Two Strings 1. File Name 2. "r" for read (for now)	File

File is a NEW Type

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

Working with Files

Preloaded Functions		
Name	Inputs	Outputs
type	Expression	Type Name
open	Two Strings 1. File Name 2. "r" for read (for now)	File

File is a NEW Type

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

Working with Files

Preloaded Functions

Name	Inputs	Outputs
type	Expression	Type
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

Name	Inputs	Outputs
type	Expression	Type
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")  
>>> fileString = myFile.read()
```

Working with Files

Preloaded Functions

Name	Inputs	Outputs
type	Expression	Type
open	Two Strings 1. File Name 2. "r" for read (for now)	File
read (On a File)	none	String
close (On a File)	none	none

```
>>> fileName = "C:\Users\Anna\Desktop\poem.txt"  
>>> myFile = open(fileName, "r")  
>>> fileString = myFile.read()  
>>> myFile.close()
```

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:\nCoffee grounds,
potato peelings,\nBrown bananas, rotten peas,\nChunks of
sour cottage cheese.\nIt filled the can, it covered the
floor,\nIt cracked the window and blocked the door\nWith
bacon rinds and chicken bones,\nDrippy ends of ice cream
cones,\nPrune pits, peach pits, orange peel,\nGloppy
glumps of cold oatmeal,\nPizza crusts and withered
greens,\nSoggy beans and tangerines,\nCrusts of black
burned buttered toast,
...
Because the hour is much too late.\nBut children, remember
Sarah Stout\nAnd always take the garbage out!'
```

- Shel Silverstein

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:\nCoffee grounds,
potato peelings,\nBrown bananas, rotten peas,\nChunks of
sou... the
fl... th
ba... am
co...
gl...
gr...
bu...
..
Because... remember
Sarah Stout\nAnd always take the garbage out!'
```

Escape Characters

'\'' means interpret the NEXT character differently.

- `\n`: “new line”
- `\'`: “apostrophe”
- `\t`: “tab”

- Shel Silverstein

Working with Files

Preloaded Functions		
Name	Inputs	Outputs
type	Expression	Type
open	Two Strings 1. File Name 2. "r" for read (for now)	File
read (On a File)	none	String
close (On a File)	none	none
split (On a String)	(optional) delimiter	List of Strings

```
>>> myList = fileString.split()
```

Working with Files

```
>>> 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

Preloaded Functions		
Name	Inputs	Outputs
type	Expression	Type
open	Two Strings 1. File Name 2. "r" for read (for now)	File
read (On a File)	none	String
close (On a File)	none	none
split (On a String)	(optional) delimiter	List of Strings

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”


```
>>> 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
>>>
```

Python For Statements (For Loops)

“For each element in list myList, do something”

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

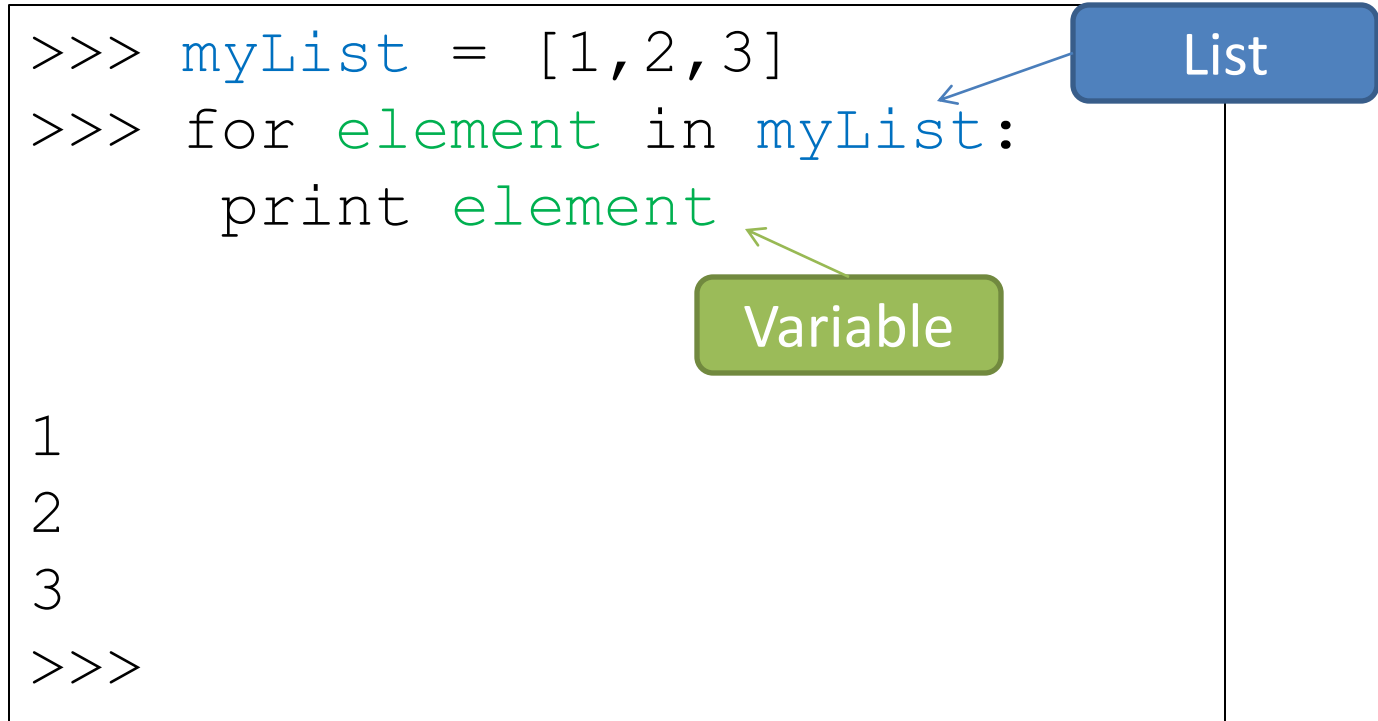
1
2
3
>>>
```



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
>>>
```



The code block is annotated with two callout boxes. A blue box labeled "List" has an arrow pointing to the variable `myList` in the first line of code. A green box labeled "Variable" has an arrow pointing to the variable `element` in the second line of code.

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
>>>
```

The diagram illustrates the components of the Python code. A blue box labeled "List" has an arrow pointing to the `myList` variable in the first line of code. A green box labeled "Variable" has an arrow pointing to the `num` variable in the `for` loop header.

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
>>>
```

The diagram illustrates the execution of a Python for loop. It shows a code block with three lines of code. The first line is `myList = [1,2,3]`, where `myList` is a list. A blue box labeled "List" points to `myList`. The second line is `for num in myList:`, where `num` is a variable. A green box labeled "Variable" points to `num`. The third line is `print num`, which is indented. A red box labeled "Indentation Matters!!" points to the indentation. The output of the code is `1`, `2`, and `3`, each on a new line. The prompt `>>>` is shown at the beginning and end of the code block.

Word Count for Shel's Poem

```
def countWordsInShel () :
```

```
    return count
```

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!

```
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

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

“Inputs” are also called *Arguments*.

Preloaded Functions

Name	Inputs	Outputs
type	Expression	Type
open	Two Strings 1. File Name 2. “r” for read (for now)	File
read (On a File)	none	String
close (On a File)	none	none
split (On a String)	(optional) delimiter	List of Strings
len	List	Integer

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

Python `For` Statements (For Loops)

“For each element in list `myList`, do something”

```
>>> 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”

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

1

2

3

>>>

Preloaded Functions

range

Two Integers

1. Start Index (Inclusive)

2. End Index (Exclusive)

List of
Integers

Python For Statements (For Loops)

“For each element in list myList, do something”

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

1
2
3
>>>

List
[0,1,2]

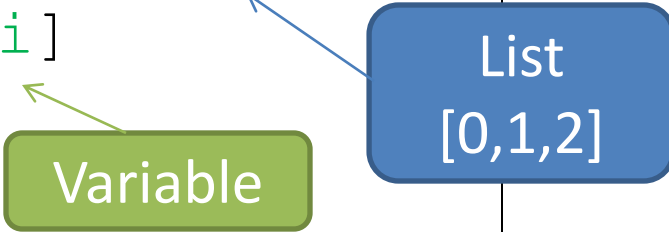
Preloaded Functions		
range	Two Integers 1. Start Index (Inclusive) 2. End Index (Exclusive)	List of Integers

The diagram illustrates the execution of a Python for loop. It shows a code snippet where a list 'myList' is defined as [1, 2, 3] and a for loop iterates over the range(0, 3). A blue callout box points to the range function, indicating it generates the list [0, 1, 2]. Below the code, a table titled 'Preloaded Functions' provides details for the 'range' function, specifying it takes two integers (start and end) and returns a list of integers.

Python For Statements (For Loops)

“For each element in list myList, do something”

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



The diagram consists of two callout boxes. A blue rounded rectangle on the right contains the text "List [0,1,2]". A blue arrow points from this box to the "range(0,3)" part of the code. A green rounded rectangle below it contains the text "Variable". A green arrow points from this box to the variable "i" in the code.

```
1
2
3
>>>
```

Python For Statements (For Loops)

“For each element in list myList, do something”

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

1
2
3
>>>

Variable

List [0,1,2]

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”

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

1
2
3
>>>

Variable

List [0,1,2]

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”

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

Preloaded Functions

<code>range</code>	Two Integers 1. Start Index (Inclusive) 2. End Index (Exclusive)	List of Integers
--------------------	--	------------------

Python For Statements (For Loops)

“For each element in list myList, do something”

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

Indentation
Matters!!

Variable

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

Returns
NOTHING!

Python Summary

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

“Inputs” are also called *Arguments*.

Preloaded Functions

Name	Inputs	Outputs
type	Expression	Type
open	Two Strings 1. File Name 2. “r” for read (for now)	File
read (On a File)	none	String
close (On a File)	none	none
split (On a String)	(optional) delimiter	List of Strings
len	List	Integer
range	Two Integers 1. Start Index (Inclusive) 2. End Index (Exclusive)	List of Integers

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.