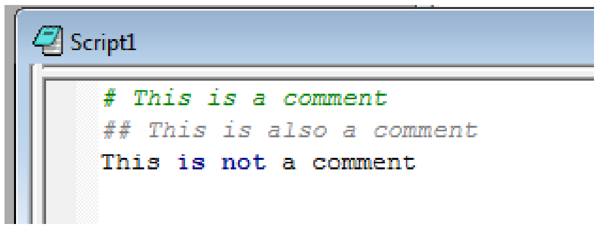
Chapter 2: Python fundamentals

2.1 Creating a Python Script file

- PythonWin offers a script window for creating, editing, running and debugging Python scripts.
- Creating a New Python Script: *On the Menu Toolbar click "**File**" →"**New**" → Select "**Python Script**" from the Dialog window → Click "**OK**"*
- Saving the python script: *Use your mouse pointer to first select the relevant script window → on the Menu Toolbar click "**File**" → "**Save As**" → browse to the location where you will like the scripts to be saved and give it a meaningful name.*

2.2 Comments

- Line of code that you add to your script that serve as documentation
- The lines are NOT executed by the Python interpreter
- A comment starts with a single pound (#) or double pound (##) sign
- Comments with beginning '#' show green and with '##'show gray in PythonWin
- Alt-3 comments an entire line, using gray text. Alt-shift-3 should undo the commenting.
- Good code should include comments to help explain what it's doing. This is helpful not only for other people reading your code, but also for the programmer understanding code later on.
- Temporarily commenting out code is handy during debugging.

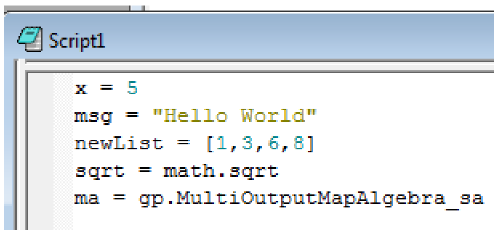


```
# This is a comment
## This is also a comment
This is not a comment
```

2.3 Variables

- Variable are used for storing values
- They can hold many different types of data; numbers, strings, lists, files, objects, geoprocessing tools in ArcGIS
- Dynamically typed: No declaration keyword or type assignment is necessary. You need only to give the variable a name and a value.

  Python: nValue = 10 C++: int nValue = 10
- Case sensitive



```
x = 5
msg = "Hello World"
newList = [1,3,6,8]
sqrt = math.sqrt
ma = gp.MultiOutputMapAlgebra_sa
```

- Geoprocessing tool names are NOT case sensitive, NOR are paths (but variable names are case sensitive). So in the last statement above,

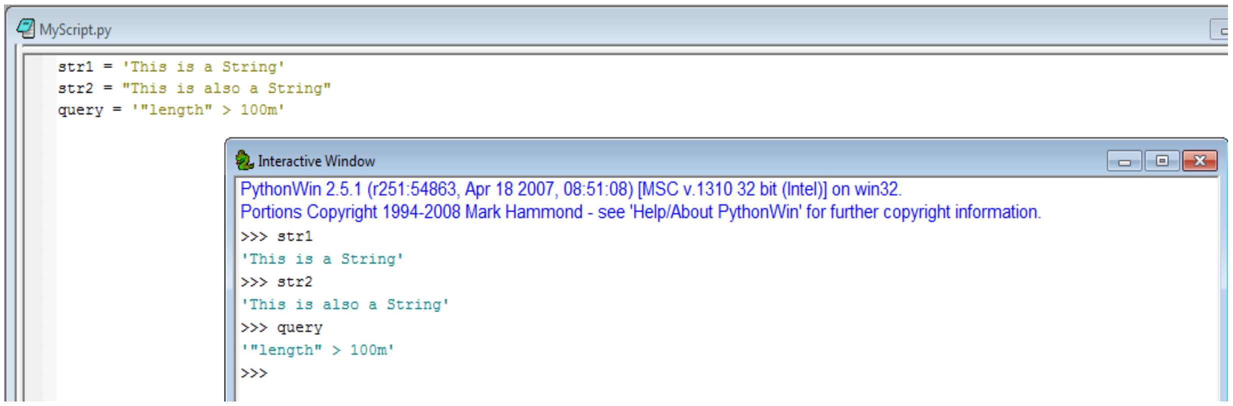
  ma = gp.MultiOutputMapAlgebra_sa

  … you could have just as well written it:

  ma = gp.MULTIOUTPUTMAPALGEBRA_SA

2.4 Strings

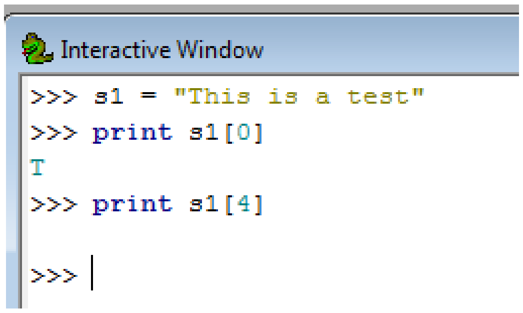
- Variables can store strings
- Strings are a collection of characters used for storing text-based information
- Defined by enclosing in single or double quotes



```
str1 = 'This is a String'
str2 = "This is also a String"
query = '"length" > 100m'
```

```
PythonWin 2.5.1 (r251:54863, Apr 18 2007, 08:51:08) [MSC v.1310 32 bit (Intel)] on win32.
Portions Copyright 1994-2008 Mark Hammond - see 'Help/About PythonWin' for further copyright information.
>>> str1
'This is a String'
>>> str2
'This is also a String'
>>> query
'"length" > 100m'
>>>
```

- Strings are indexed (zero based).



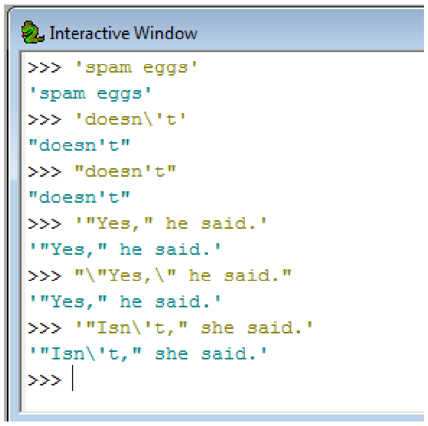
```
>>> s1 = "This is a test"
>>> print s1[0]
T
>>> print s1[4]

>>>
```

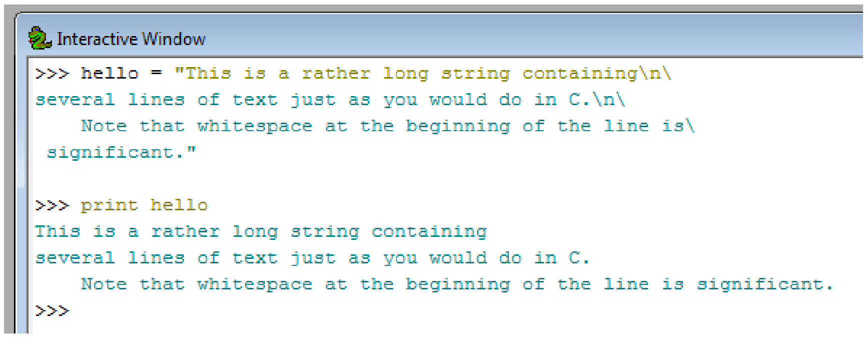
- Reverse indexing starts with the last character position being ‘-1’
- You CANNOT use string indexing for assignment and it can only be used to extract a single value.

- For special characters, use the escape character '\'

\n	newline
\t	tab
\\	'\'
\	line continuation

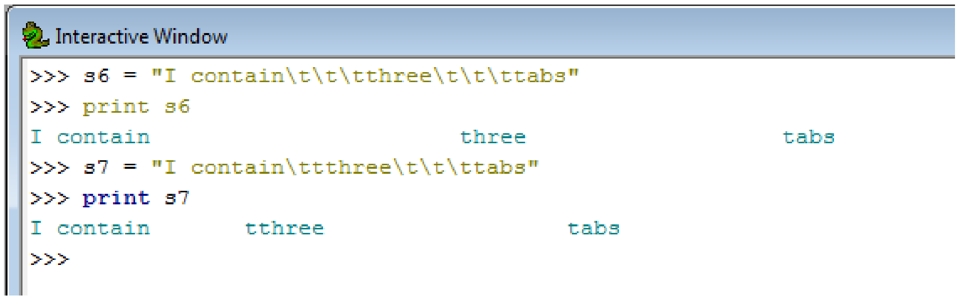


```
>>> 'spam eggs'
'spam eggs'
>>> 'doesn\'t'
"doesn't"
>>> "doesn't"
"doesn't"
>>> '"Yes," he said.'
'"Yes," he said.'
>>> "\"Yes,\" he said."
'"Yes," he said.'
>>> '"Isn\'t," she said.'
'"Isn\'t," she said.'
>>> 
```



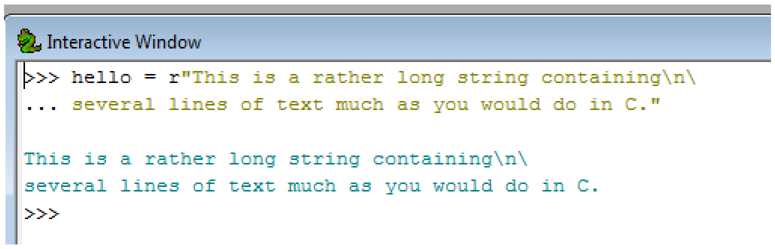
```
>>> hello = "This is a rather long string containing\n\
several lines of text just as you would do in C.\n\
    Note that whitespace at the beginning of the line is\
 significant."

>>> print hello
This is a rather long string containing
several lines of text just as you would do in C.
    Note that whitespace at the beginning of the line is significant.
>>>
```



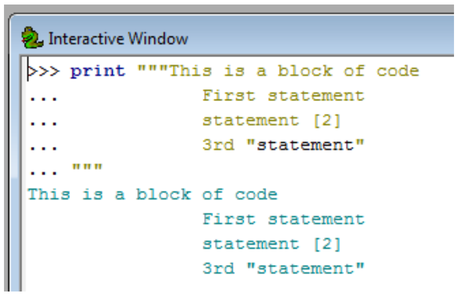
```
>>> s6 = "I contain\t\t\tthree\t\t\ttabs"
>>> print s6
I contain                        three                       tabs
>>> s7 = "I contain\tttthree\t\t\ttabs"
>>> print s7
I contain       tthree                   tabs
>>>
```

- The print command removed the quotations from the output
- If we make the string a ``raw'' string, the `\n` sequences are not converted to newlines, but the backslash at the end of the line, and the newline character in the source, are both included in the string as data.

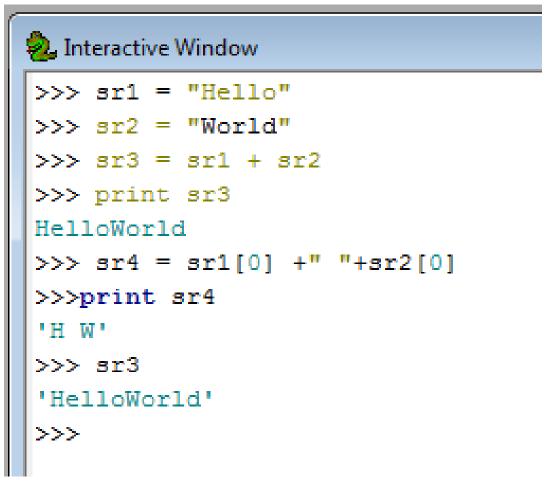

```
>>> hello = r"This is a rather long string containing\n\
... several lines of text much as you would do in C."

This is a rather long string containing\n\
several lines of text much as you would do in C.
>>>
```

- Strings can be surrounded in a pair of matching triple-quotes: """ or '''.


```
>>> print """This is a block of code
...             First statement
...             statement [2]
...             3rd "statement"
... """
This is a block of code
                First statement
                statement [2]
                3rd "statement"
```

- Strings can be concatenated using the '+' operator


```
>>> sr1 = "Hello"
>>> sr2 = "World"
>>> sr3 = sr1 + sr2
>>> print sr3
HelloWorld
>>> sr4 = sr1[0] +" "+sr2[0]
>>>print sr4
'H W'
>>> sr3
'HelloWorld'
>>>
```

- Pathnames are a special case of strings. Because the backslash '\' represents a special escape character typical path names using single backslashes will not work.
- Pathnames defined as strings use two backslashes '\\' or single forward slashes '/'

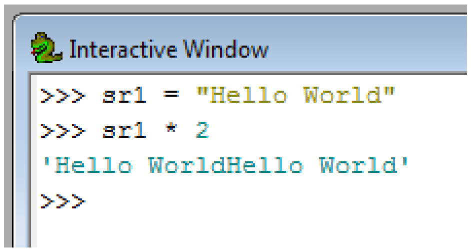
"C:\\ProgramFiles\\Windows\\Python\\Counties.shp"

Or

"C:/ProgramFiles/Windows/Python/Counties.shp"

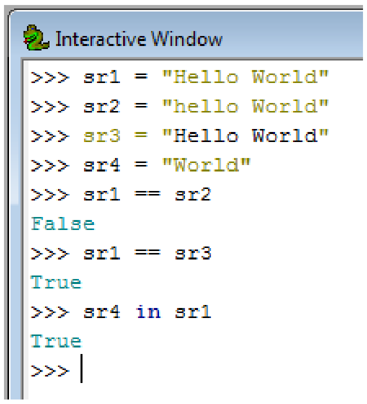
- Use can also use string literal with a letter r before the string
  r"C:\ProgramFiles\Windows\Python\Counties.shp"

- An empty string is defined by two quotes “” or ‘’ with nothing in between.
- String repeating is accomplished using the ‘*’ operator


Interactive Window

```
>>> sr1 = "Hello World"
>>> sr1 * 2
'Hello WorldHello World'
>>>
```

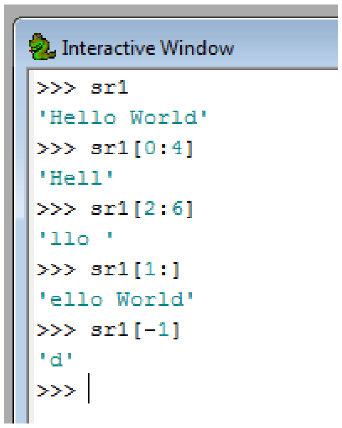
- Strings can be assigned using the ‘=’ operator
- Strings can be tested for equality using the ‘==’ operator
- Strings can be tested for containment using the ‘in’ operator


Interactive Window

```
>>> sr1 = "Hello World"
>>> sr2 = "hello World"
>>> sr3 = "Hello World"
>>> sr4 = "World"
>>> sr1 == sr2
False
>>> sr1 == sr3
True
>>> sr4 in sr1
True
>>>
```

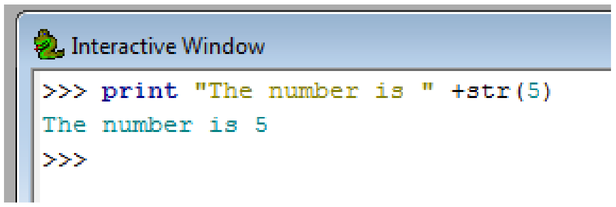
- String slicing is use to extract a CONTIGUOUS sequence of string literals

  stringName [StartPosition:StopPosition]


Interactive Window

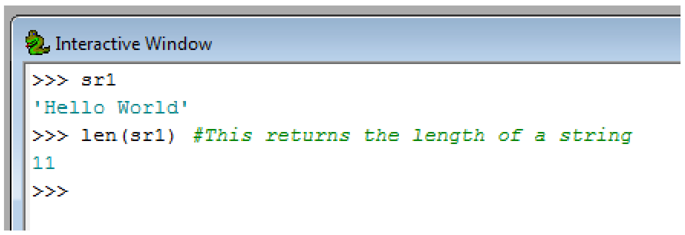
```
>>> sr1
'Hello World'
>>> sr1[0:4]
'Hell'
>>> sr1[2:6]
'llo '
>>> sr1[1:]
'ello World'
>>> sr1[-1]
'd'
>>>
```

- Type casting is used to join strings to non string values



```
>>> print "The number is " +str(5)
The number is 5
>>>
```

- There are many string functions bundled within every python installation



```
>>> sr1
'Hello World'
>>> len(sr1) #This returns the length of a string
11
>>>
```

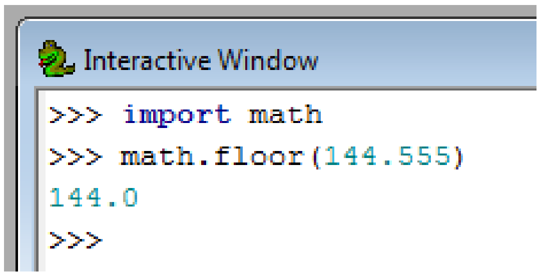
- For more functions, see section 3.6.1 in the user documentation

2.5 Numbers

- Python can also store numbers which are numeric data types; plain integers (int), long integers (long), floating point numbers (float), and imaginary numbers (complex)
- Numbers DO NOT contain quotes and MUST BE a numeric value
- Most numeric operations are familiar

Operation	Result
`x + y`	sum of *x* and *y*
`x - y`	difference of *x* and *y*
`x * y`	product of *x* and *y*
`x / y`	quotient of *x* and *y*
`x // y`	(floored) quotient of *x* and *y*
`x % y`	remainder of `x / y`
`-x`	*x* negated
`+x`	*x* unchanged
`abs(x)`	absolute value or magnitude of *x*
`int(x)`	*x* converted to integer
`long(x)`	*x* converted to long integer
`float(x)`	*x* converted to floating point
`complex(re, im)`	a complex number with real part *re*, imaginary part *im*. *im* defaults to zero.
`c.conjugate()`	conjugate of the complex number *c*
`divmod(x, y)`	the pair `(x // y, x % y)`
`pow(x, y)`	*x* to the power *y*
`x ** y`	*x* to the power *y*

- Additional mathematical functions are contained within the Math Module which first must be imported.

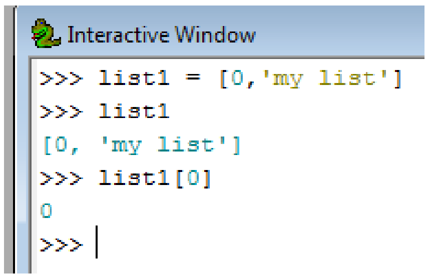


```
>>> import math
>>> math.floor(144.555)
144.0
>>>
```

- Most results are reported as real numbers. See section 6.1 of user documentation for available functions.

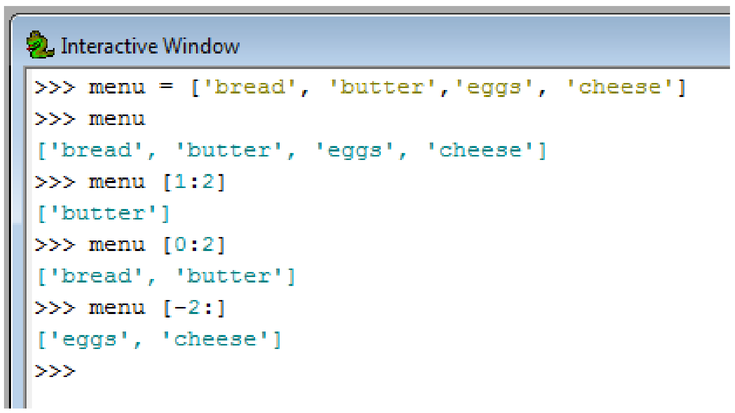
2.6 Lists

- Lists stores sets of related data in an ordered fashion. Data types include strings, numbers, other lists or even objects.
- They are zero-based
- Can grow and shrink
- Lists are created similarly to strings with the exception that many data types can be stored and data are separated by commas.
- They can also be accessed using indexing similar to that of strings.



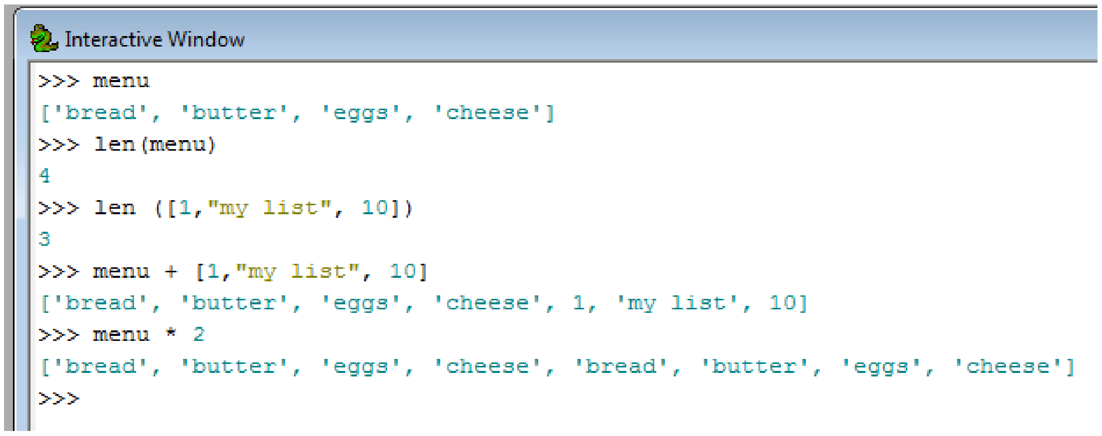
```
>>> list1 = [0,'my list']
>>> list1
[0, 'my list']
>>> list1[0]
0
>>> 
```

- Slicing can also be used to access contiguous values



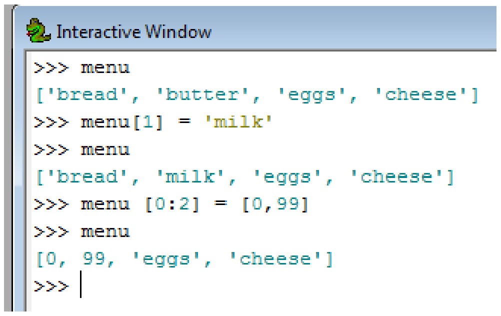
```
>>> menu = ['bread', 'butter','eggs', 'cheese']
>>> menu
['bread', 'butter', 'eggs', 'cheese']
>>> menu [1:2]
['butter']
>>> menu [0:2]
['bread', 'butter']
>>> menu [-2:]
['eggs', 'cheese']
>>>
```

- Common list operations include finding the length, concatenation and repetition of lists



```
>>> menu
['bread', 'butter', 'eggs', 'cheese']
>>> len(menu)
4
>>> len ([1,"my list", 10])
3
>>> menu + [1,"my list", 10]
['bread', 'butter', 'eggs', 'cheese', 1, 'my list', 10]
>>> menu * 2
['bread', 'butter', 'eggs', 'cheese', 'bread', 'butter', 'eggs', 'cheese']
>>>
```

- Unlike strings, the contents of a list can be changed dynamically. Indexing is used to change a single value while slicing is used to change a sequence of values.



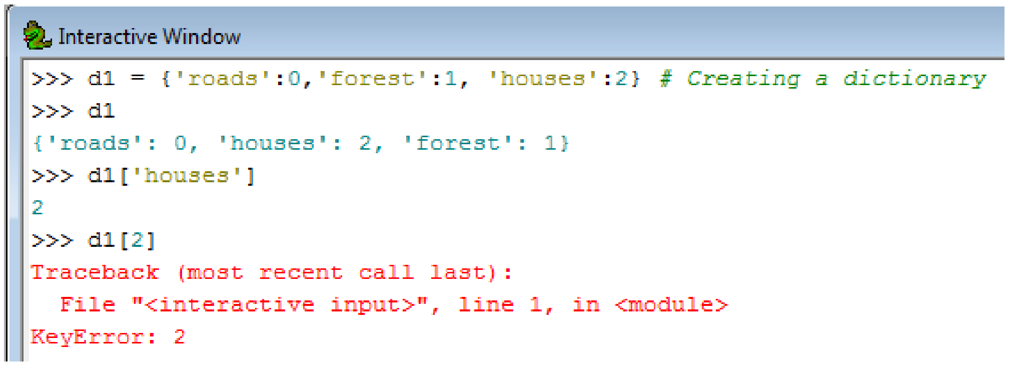
```
>>> menu
['bread', 'butter', 'eggs', 'cheese']
>>> menu[1] = 'milk'
>>> menu
['bread', 'milk', 'eggs', 'cheese']
>>> menu [0:2] = [0,99]
>>> menu
[0, 99, 'eggs', 'cheese']
>>>
```

- See section 5 of Python Tutorial for a list of common list methods along with options for using lists as stacks and queues.

2.7 Dictionaries

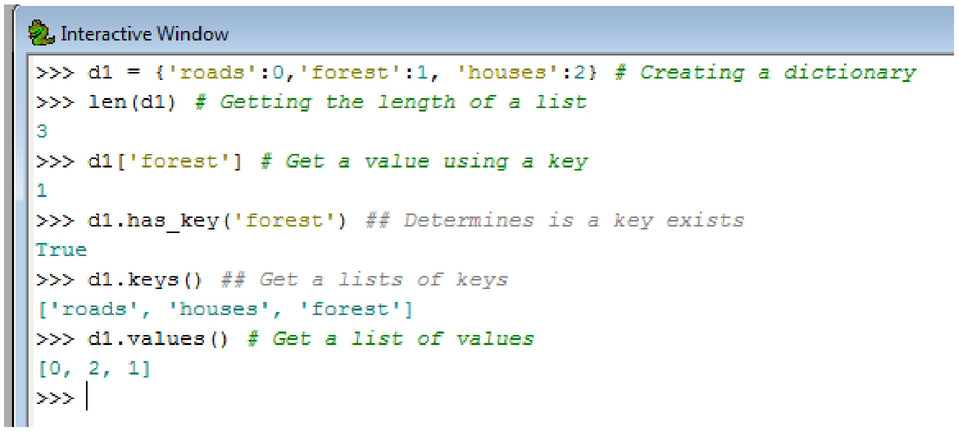
- Store a collection of data similar to list, however, unlike lists, dictionary contain UNORDERED values.
- Dictionary values can be accessed by keys

  Key : value



```
>>> d1 = {'roads':0,'forest':1, 'houses':2} # Creating a dictionary
>>> d1
{'roads': 0, 'houses': 2, 'forest': 1}
>>> d1['houses']
2
>>> d1[2]
Traceback (most recent call last):
  File "<interactive input>", line 1, in <module>
KeyError: 2
```

- Basic operations include getting the number of items in a dictionary, retrieving a value using a key and determining if a key exists.



```
>>> d1 = {'roads':0,'forest':1, 'houses':2} # Creating a dictionary
>>> len(d1) # Getting the length of a list
3
>>> d1['forest'] # Get a value using a key
1
>>> d1.has_key('forest') ## Determines is a key exists
True
>>> d1.keys() ## Get a lists of keys
['roads', 'houses', 'forest']
>>> d1.values() # Get a list of values
[0, 2, 1]
>>> |
```

- Slicing DOES NOT work on lists since they are unordered.

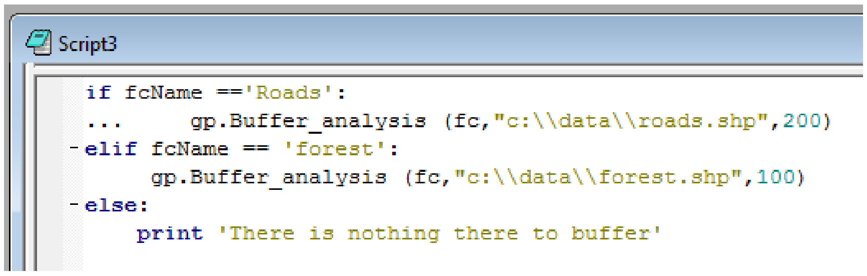
2.8 Statements

- Represents each line in a script OTHER THAN a comment
  - Assignment statement
    x = 5
  - Print statement
    print x
  - Import statement
    import win32com.client
  - Geoprocessing statement
    gp.HillShade_sa (elev, hillsh, azimuth, sunangle)
  - Decision statement
    if x>5:
  - Loop statements
    while x < 5:
    for i in range(5):

Decision statements

- If…elif…else tests for a true condition
- Enables you to control the flow of your program and make decisions.
  - If a point feature class exists in a polygon feature class, gets its' XY coordinates
  - If the name of the feature class is equal to road, gets its length attribute.
- Syntax
  - Colons at the end of each condition
  - Two equal signs to test for conditions
  - Indentation required
  - To finish a condition, un-indent your code (Python indents automatically)

```
if <condition1>:
    <Statements>
elif <condition2>:
    <Statements>
else:
    <Statements>
```



Script3

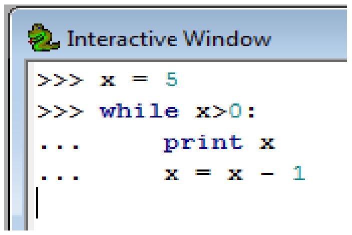
```
if fcName =='Roads':
...     gp.Buffer_analysis (fc,"c:\\data\\roads.shp",200)
elif fcName == 'forest':
     gp.Buffer_analysis (fc,"c:\\data\\forest.shp",100)
else:
    print 'There is nothing there to buffer'
```

Syntax rules

- Statements are executed sequentially unless interrupted by code branches (decision and loop statements)
- Python detects statement and block boundaries automatically.
  - No braces or delimiters around blocks of code
  - Indentation is used to group statements in a block
  - Statements are not terminated with a semicolon; end of line character marks the end of a statement.
- Compound statements include a ':' character
  - Compound statements follow the pattern: header line terminated by a colon
  - Next statements follow an indented pattern. These are called blocks
- Python ignores all spaces and comments.
- All code blocks indented the same distance belong to the same block of code until that block is ended by a line less indented.

Looping statements

- Allows repetition of lines of code
- While loops repeatedly execute a block of statements while a test at the top of the loop evaluates to true.



```
>>> x = 5
>>> while x>0:
...     print x
...     x = x - 1
```

- For loops execute a block of statements a predetermined number of times
- For loops can be one of two kinds; counted loops and list loops
- Counted loops:
  - Increment and test a variable on each iteration of the loop
  - When the variable reaches a pre-determined number the loop exists
  - Use range function

Generates lists containing arithmetic progressions:

```
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

It is possible to let the range start at another number, or to specify a different increment.

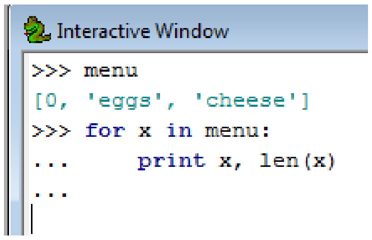
```
>>> range(5, 10)
[5, 6, 7, 8, 9]
>>> range(0, 10, 3)
[0, 3, 6, 9]
>>> range(-10, -100, -30)
[-10, -40, -70]
```

To iterate over the indices of a sequence, combine range() and len() as follows:

```
>>> a = ['Mary', 'had', 'a', 'little', 'lamb']
>>> for i in range(len(a)):
...     print i, a[i]
...
0 Mary
1 had
2 a
```

3 little
4 lamb

- List loops
    - Iterate over each value in a list
    - The loop executes once for each value and then stops looping



```
>>> menu
[0, 'eggs', 'cheese']
>>> for x in menu:
...     print x, len(x)
...
```

Breaking and continuing loops

- Break-jumps out of the closest enclosing loop
- Continue-jumps to the top if the closest enclosing loop
- Syntax

```
While <test>:
        <statements>
        If <test>: break  # exist loop now
        If <test>: continue #go to the top of the loop
```

2.9 File Input/Output

Opening files

- The open function creates a Python file object, which serves as a link to a file residing on your computer.
- Reading

    f = open (‘c:\\data\\data.txt”,’r’)
- Writing-overwrites all existing information

    f = open (‘c:\\data\\data.txt”,’w’)
- Append-adds data to the end of the file

    f = open (‘c:\\data\\data.txt”,’a’)

Closing files

- After a file has been read or written to and is no longer being used it is always a good idea to close the file.

  ```
  f.close()
  ```

Reading data

- Reading one line at a time into a string variable

  ```
  Line = f.readline()
  ```
- Read all the contents of the file into a string variable until the end of file character.

  ```
  Line = f.read()
  ```
- Read until EOF using `readline()` and return a list containing the lines thus read.

  ```
  Line = f.readlines()
  ```

2.9.4 Writing data

- Writing single string arguments

  ```
  Outfile.write ("The file must be opened first before it can be written to")
  ```
- Writing out the contents of a list structure to a file

  ```
  Menu = ["eggs","cheese","butter","milk"]
  Outfile.writelines(Menu)
  ```

2.10 Modules

- Functions that can be leveraged to assist in simplifying code
- They are usually bundled into related classes. E.g. Math module
- Python comes with a large number of built-in modules
- Modules can also be created by the user.
- Modules must be imported before they can be used

  ```
  Import math
  Import string
  ```

2.11 Exercise 1