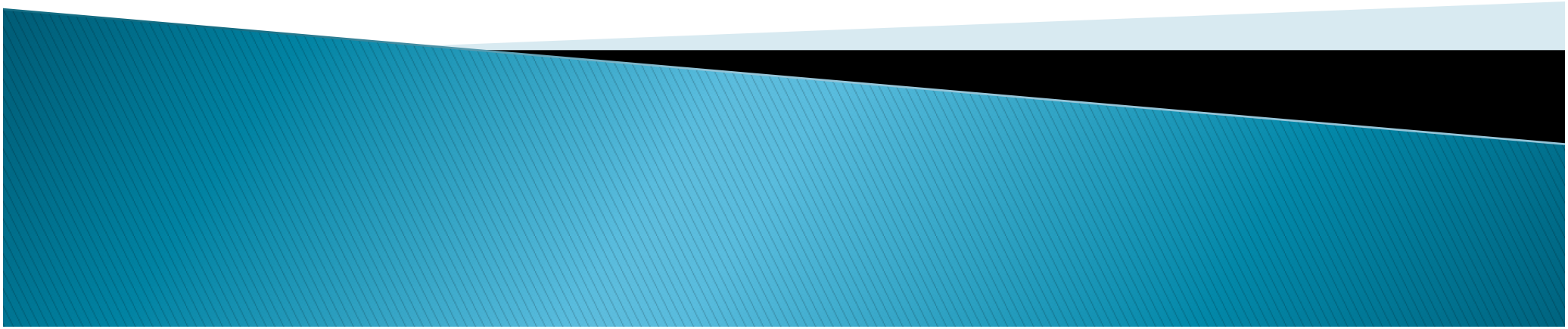
CSCI-UA.60-1 Database Design and Implementation

Using Python to "scrub" data files:

Python Review and Introduction to Python File I/O

Prof Deena Engel
Department of Computer Science
deena.engel@nyu.edu

First … a quick review on Strings …



String Indexing

- Strings are indexed; that means that every character in the string (even blank spaces) have a position number. These numbers increase in order by 1, starting from the left-most position at zero
- Notice that one can also reference every character in the string by a negative number, which starts counting from the right. To avoid confusion, when we start counting with -1 (rather than zero) when counting from the right:

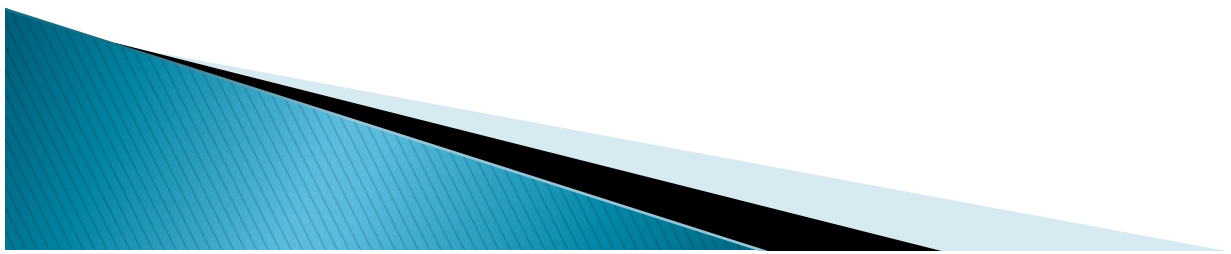
Examples of string indexing:


```
s1 = "Shakespeare"
>>>  s1[0]
'S'
>>> s1[-1]
'e'
>>> s1[5]
's'
>>> s1[-6]
's'
```

Slicing Strings

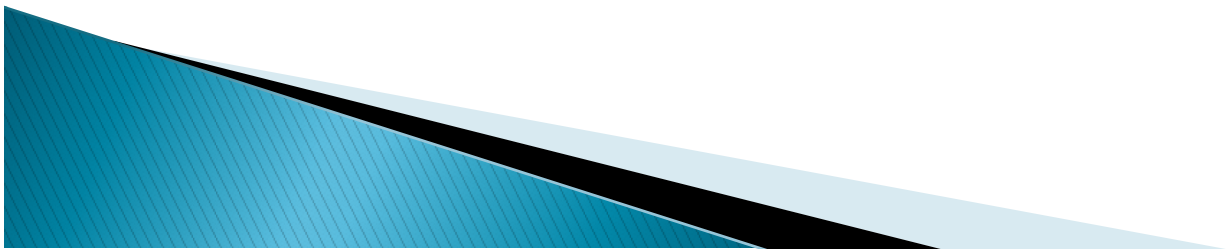
- Substrings are one or more consecutive characters within a string.
- The first location is always zero and the last position is one less than the length.
- Slicing follows this pattern:

text[begin:end:increment]



Characters and Strings

```
>>> w = 'cat'
>>> len(w)
3
>>> w[0]
'c'
>>> w[1]
'a'
>>> w[2]
't'
>>> w[0:3]
'cat'
>>>
```

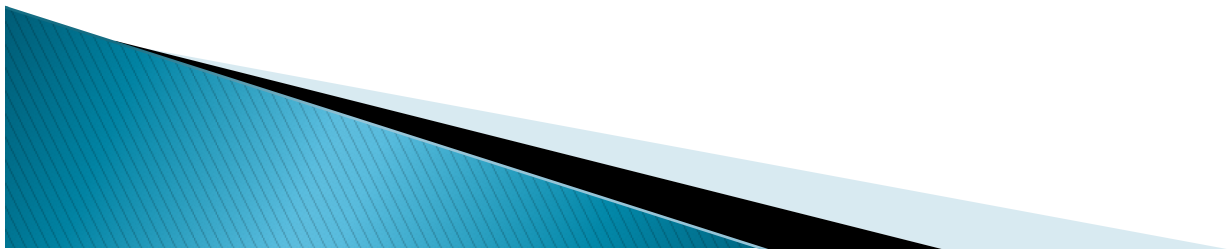


Shortcuts

```
>>> s = 'peanut butter'
>>> s[0:6]
'peanut'
>>> s[0:7]
'peanut ‘
>>> s[7:]
'butter'
>>> s[6]
' '
>>> s[:6]
'peanut‘
>>> s[7:13]
'butter'
```

Using negative index numbers:

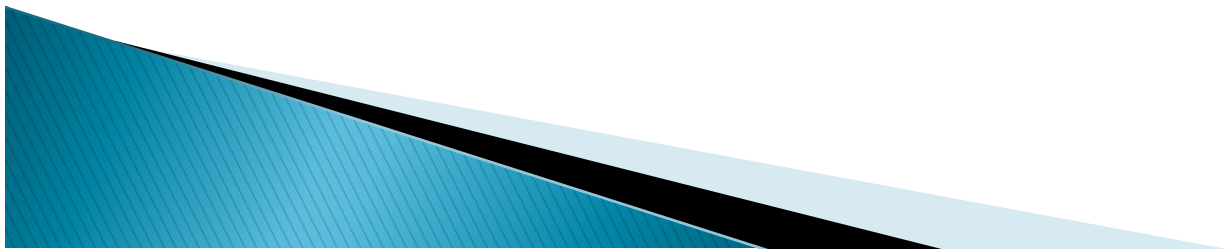
```
>>> s2 = 'butterfly'
>>> len(s2)
9
>>> s2[-1:]
'y'
>>> s2[-9:]
'butterfly'
>>> s2[-9:-3]
'butter'
>>>
```



Using the keyword “in”

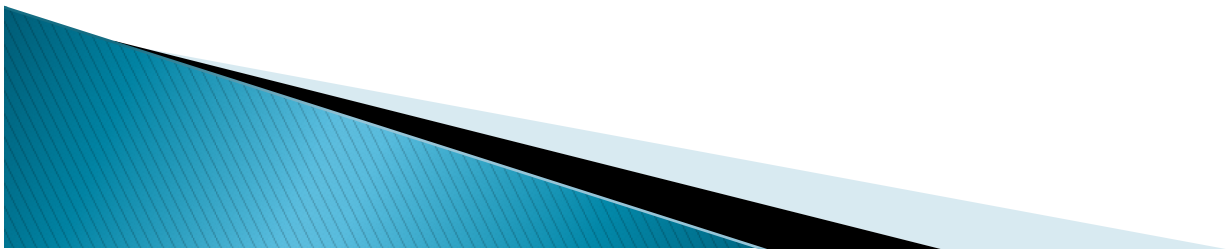
- Use *in* to see if a letter or letter combination are included in the string you wish to examine:

```
>>> p = 'pineapple'
>>> 'pine' in p
True
>>> 'apple' in p
True
>>> 'cherry' in p
False
```



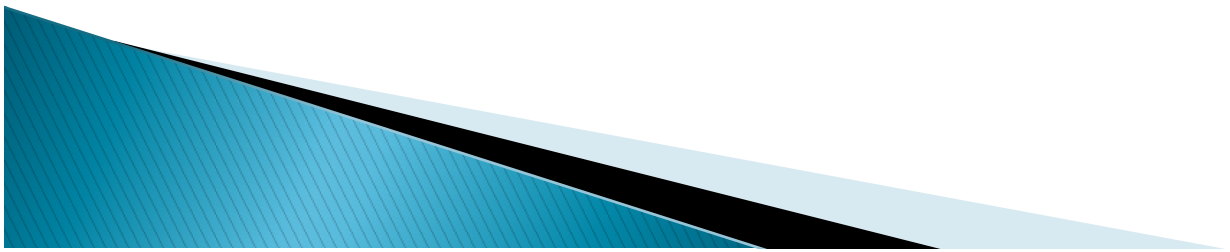
Some standard string-testing methods in Python:

- for a given string *s* …
  - s.isalpha() – s contains only letters
  - s.islower() – s contains only lower-case
  - s.isnumeric() – s contains only numeric characters
  - s.isprintable() – contains only printable characters
  - s.isupper() – contains only upper-case



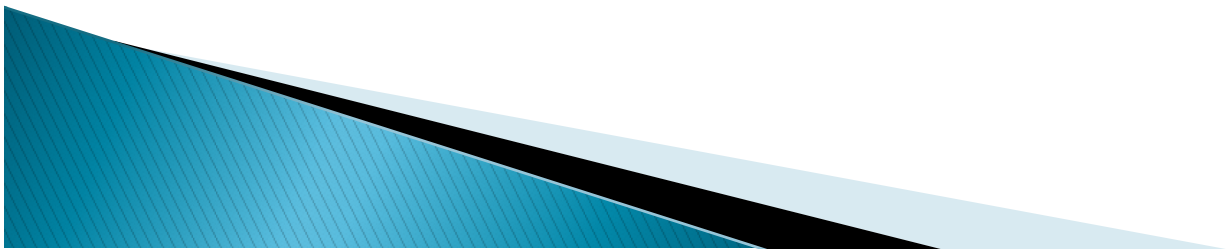
Searching strings

- for given strings *s* and *t* …
  - s.find(t) – -1 or the index where t starts in s
  - s.rfind(t) – same as find but searches from right to left
  - s.index(t) – same as find, but raises an error if t is not in s
  - s.rindex(t) – same as index, but searches from right to left



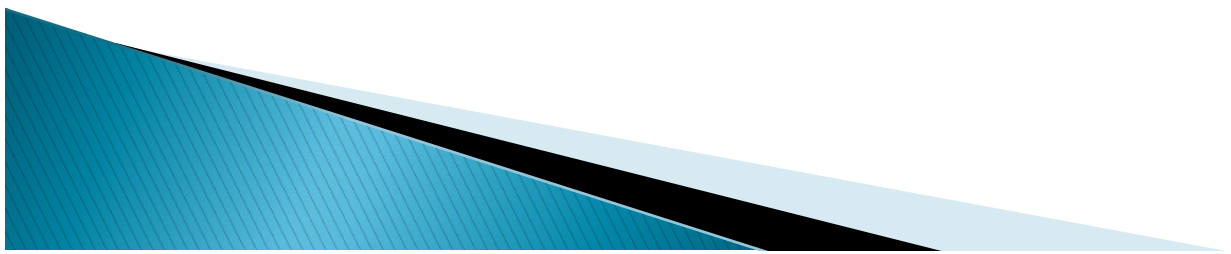
To change the case of a string using a method:

- for a given string, *s* …
  - s.capitalize – s[0] is rendered in upper-case
  - s.lower() – all letters in s are set to lower-case
  - s.upper() – all letters in s are set to upper-case
  - s.swapcase() – all lowercase letters are capitalized and vice-versa



String-stripping methods

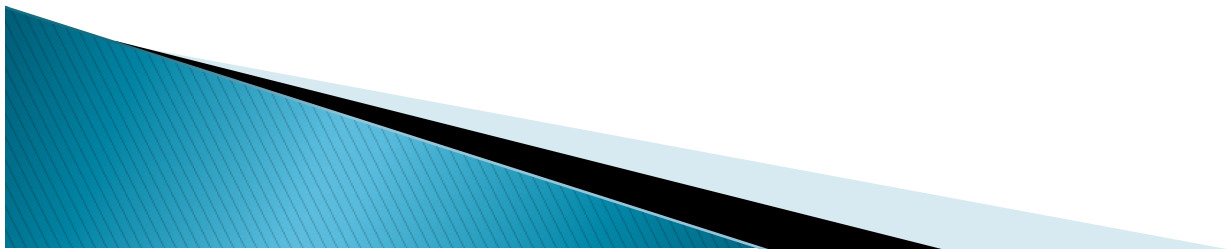
- for a given string, *s* …
  - s.strip(ch) – removes all ch characters in t occurring at the beginning or end of s
  - s.lstrip(ch) Same for characters at the left-hand side
  - s.rstrip(ch) same for characters at the right-hand side



Other useful string methods

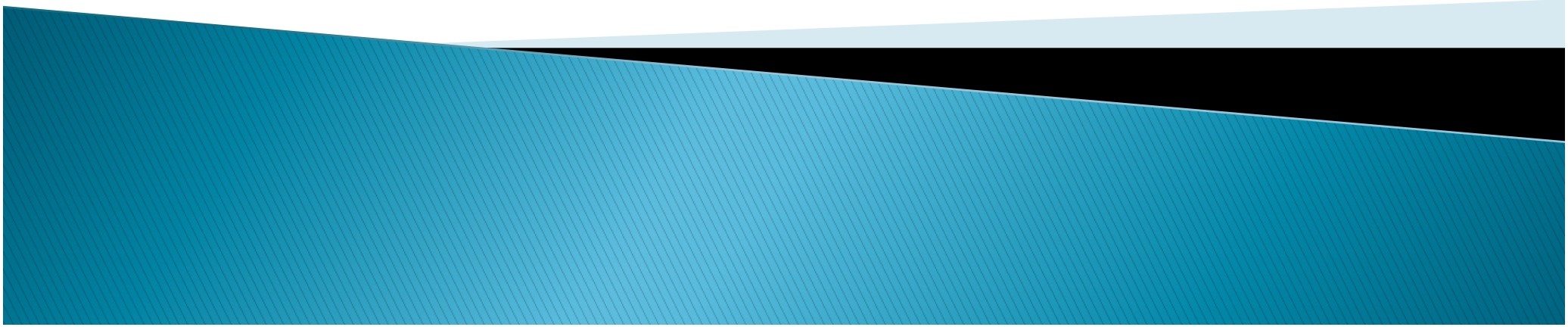
- for a given string *s* …
  - s.split(t) – returns a list of substrings in s that are separated by t
  - s.join(*seq*) – concatenates strings in a list or sequence using s as a separator

```
>>> ' -'.join(['Winnie','the','Pooh'])
Winnie-the-Pooh
```

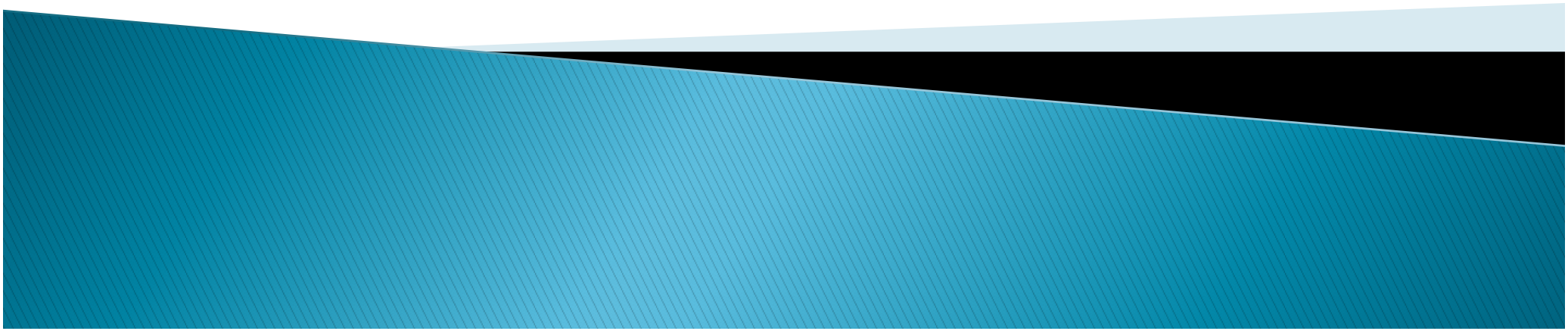


These are some of the many useful ways to work with strings … there are many more for you to explore as needed.

Python documentation on strings for further reading:
http://docs.python.org/2/library/string.html



Next … a review of lists…



Lists

There are a number of similarities between lists and strings; some of them are:

1. You can use the function len() to find out the length of a list
2. You can concatenate lists using the + symbol
3. You can repeat lists using the * symbol
4. You can use indexing to examine a specific item within a list
5. You can use slicing to examine a sublist

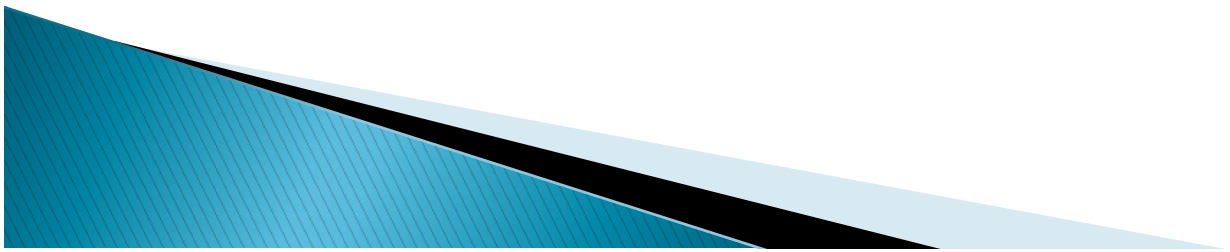
More about lists:

1. An empty list is denoted by [ ]
2. A singleton list contains only one element: [x]

Lists – continued

Lists can be changed (that is lists are *mutable*):

```
>>> colors = ['red', 'blue', 'yellow']
>>> colors
['red', 'blue', 'yellow']
>>> colors[2] = 'green'
>>> colors
['red', 'blue', 'green']
```



Working with Lists

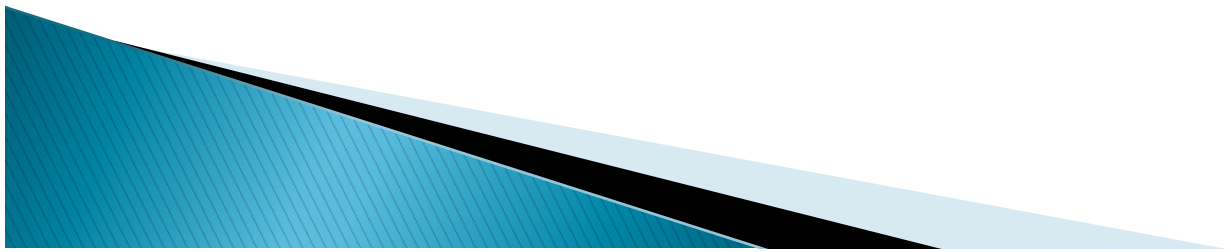
Useful for working with lists:

x.append(y) – appends y to the list x
x.sort() – sorts the elements in x
x.reverse() – reverses the order of the elements in x
x.insert(position, value) – inserts a new value at in the indicated position

To copy a list and keep a copy of the original list, we need to "clone" it:

```
>>> flowers = ['tulip', 'rose', 'buttercup']
>>> flowers
['tulip', 'rose', 'buttercup']
>>> flowers2 = flowers[:]
>>> flowers2
['tulip', 'rose', 'buttercup']
```

Notice that to clone the list, we effectively copy each element.



Working with Lists: List Comprehensions

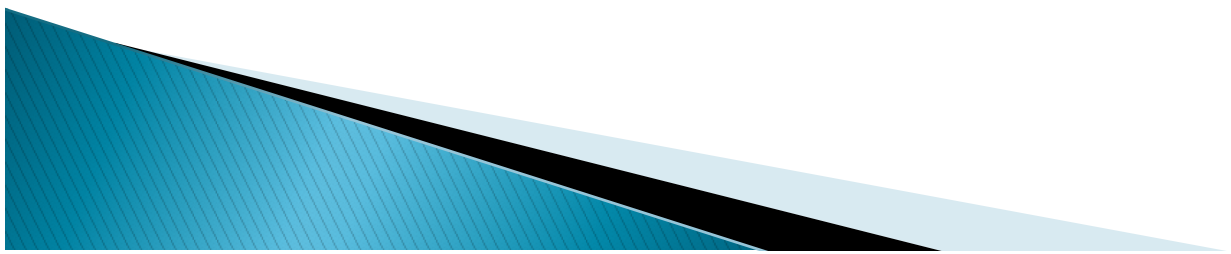
List comprehensions can be very useful for data management.

List comprehensions provide a practical and efficient way to create lists.

The result is a new list that results from evaluating the for and if clauses used in context.

Syntax:

```
new_list = [ expression for item in list if conditional ]
```

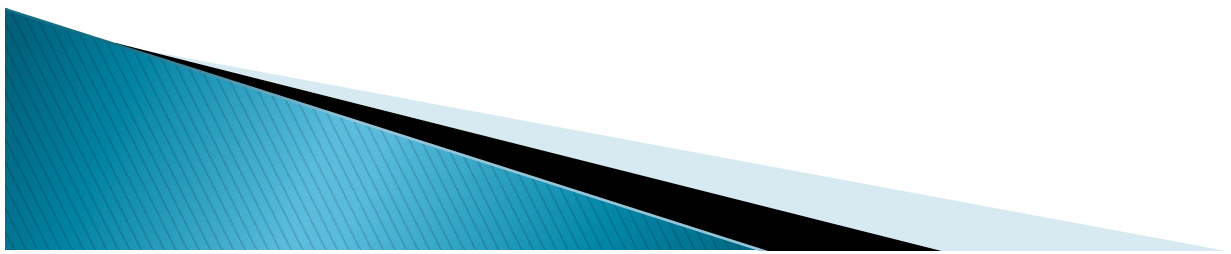


Working with Lists: List Comprehension Examples

```
# to create a new list of squares from 0-9:
>>> squares = [x**2 for x in range(10)]
>>> squares
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

# to create a new list containing the lengths of all of the words in a list:
>>> sentence = 'Call me Ishmael'
>>> s = sentence.split()
>>> s
['Call', 'me', 'Ishmael']
>>> lengths = [ len(x) for x in s ]
>>> lengths
[4, 2, 7]

# to identify only the long words (in this case, with a length over 2 characters) in a
   list:
>>> long_words = [ x for x in s if len(x)>2]
>>> long_words
['Call', 'Ishmael']
>>>
```



Working with Lists: Python documentation

Python documentation: an introduction to lists:

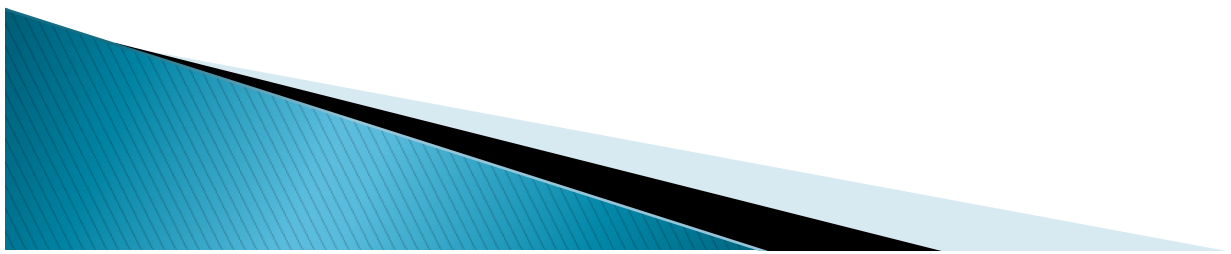
http://docs.python.org/3.1/tutorial/introduction.html#lists

Python documentation on methods for working with lists:

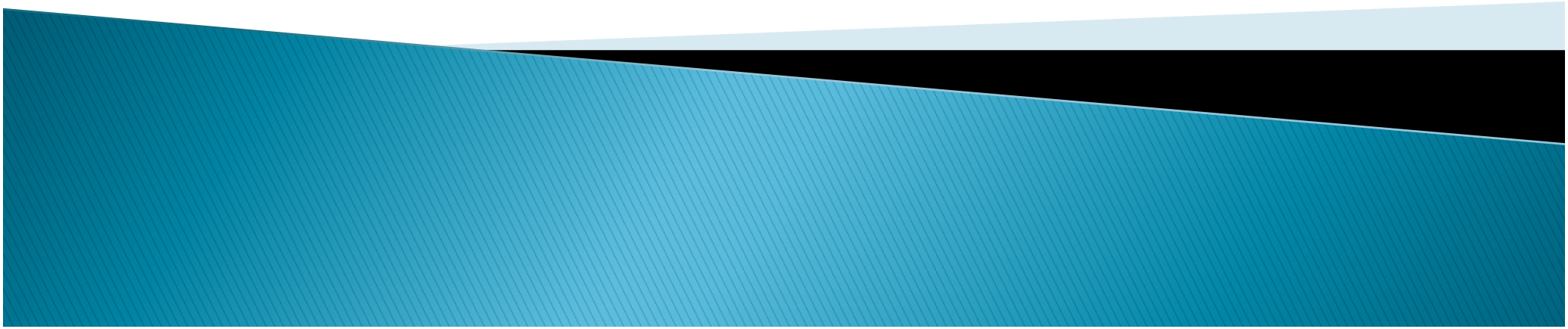
http://docs.python.org/3.1/tutorial/datastructures.html

Python documentation on list comprehensions:

https://docs.python.org/3/tutorial/datastructures.html#list-comprehensions

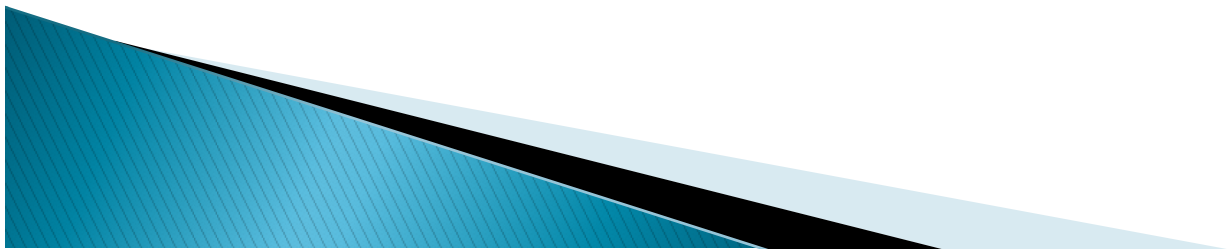


Python:
Dictionaries Review



Lists …

- As we have seen, lists use integers to index the items within a given list
  - `>>> c = ['red','blue','green']`
  - `>>> c[0]`
  - `'red'`
  - `>>> c[1]`
  - `'blue'`
  - `>>> c[2]`
  - `'green'`

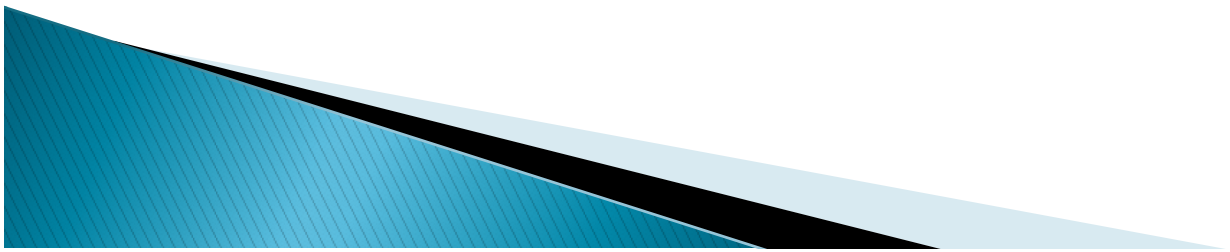


Dictionaries in Python

- Dictionaries use key:value pairs.
- In other words, instead of using *integers* for the index keys as lists do, dictionaries use *strings* for the index keys.
- Syntax:
  - lists use square brackets:

    […]
  - dictionaries use curly braces:

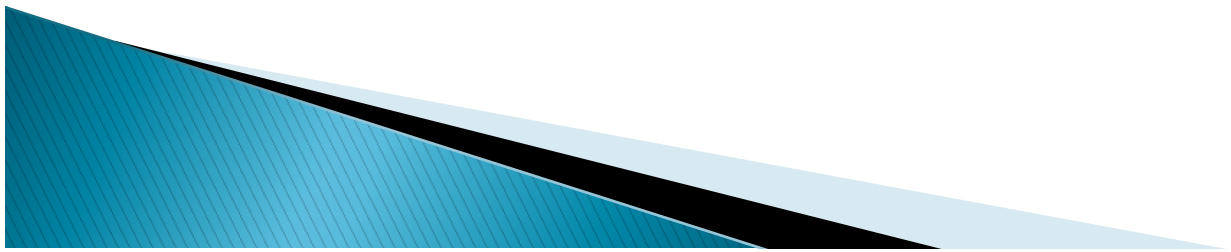
    { … }



Counting to three in French using a list:

- Using a list, we could semantically associate the words in French for "one", "two" and "three" with the integer index as follows:

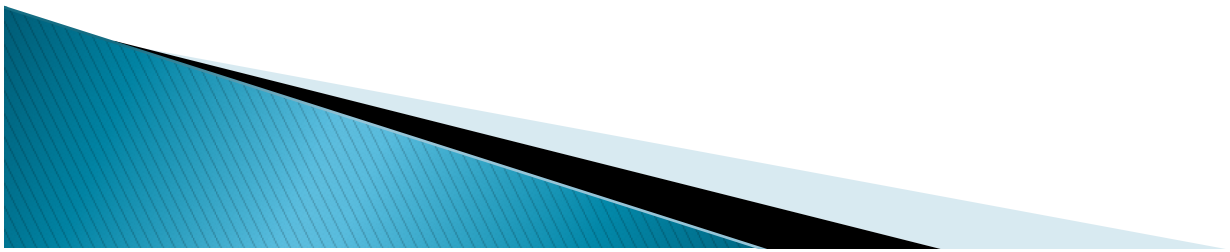
```
>>> countFrenchList = ['','un','deux','trois']
>>> countFrenchList
['', 'un', 'deux', 'trois']
>>> countFrenchList[2]
'deux‘
```



Counting to three in French using a python dictionary:

- Using a dictionary, we can associate each word in French with an English equivalent:

```
>>> countFrenchDict = { "one":"un",
  "two":"deux","three":"trois" }
>>> countFrenchDict
{'three': 'trois', 'two': 'deux', 'one': 'un'}
>>> countFrenchDict['two']
'deux`
```



Counting to four in French using a python dictionary:

- Note the syntax of a key:value pair:

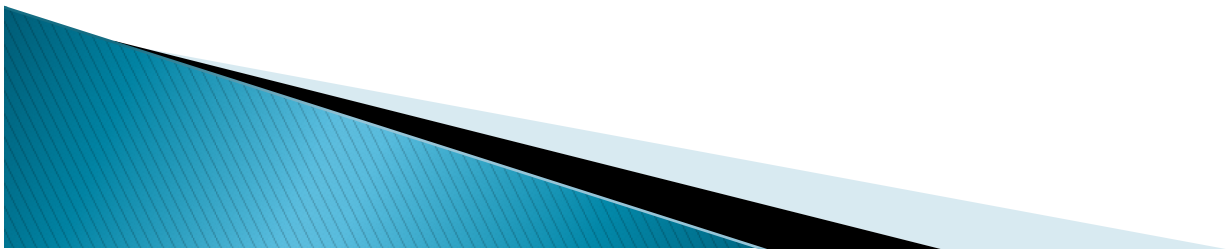
```
'two':'deux'
```

- … and how to reference a data element in the dictionary:

```
>>> countFrenchDict['two']
'deux'
```

- If you want to add data to a dictionary, simply provide the key and the value:

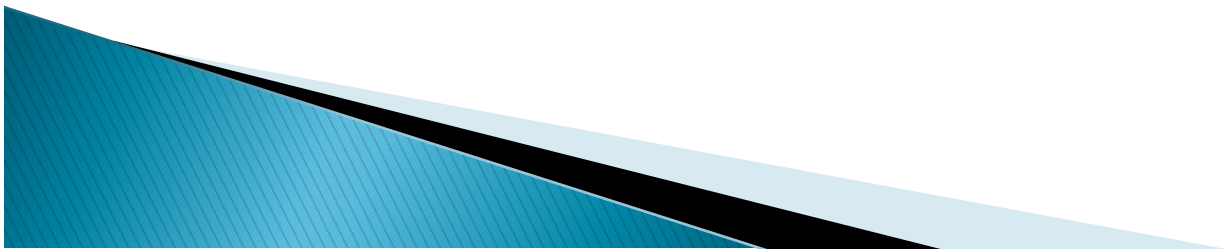
```
>>> countFrenchDict['four'] = "quatre"
```



More on dictionaries

- The order of the elements in a dictionary is not predictable; however, python has built-on algorithms to make it run very quickly.
- In fact, the order of the elements in a dictionary does not matter because python looks up elements by the key.

```
>>> print(countFrenchDict['three'])
trois
```



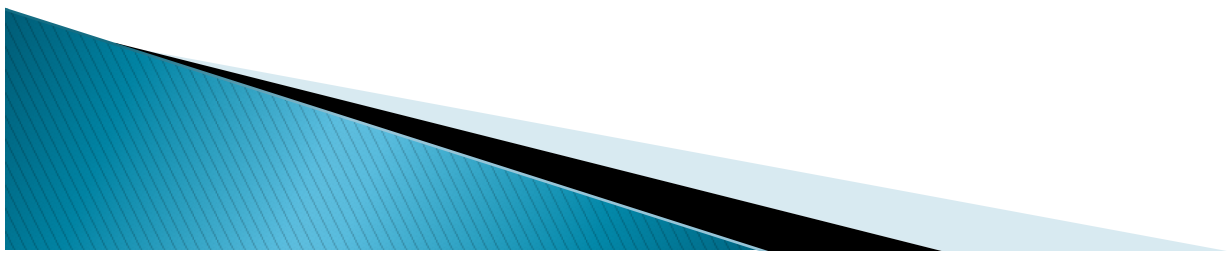
Dictionary Operations

- To find out the length of a dictionary:

```
>>> len(countFrenchDict)
4
```

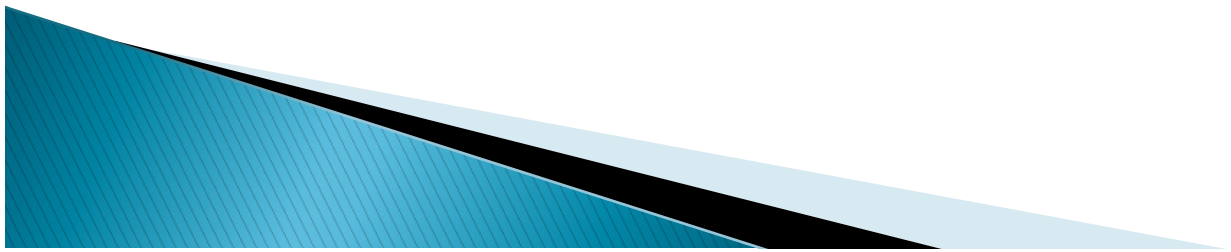
- Use in and not in to see if an element is in a dictionary. Note that in and not in look up keys, not values:

```
>>> 'two' in countFrenchDict
True
>>> 'deux' in countFrenchDict
False
```



Dictionaries and Working with Data

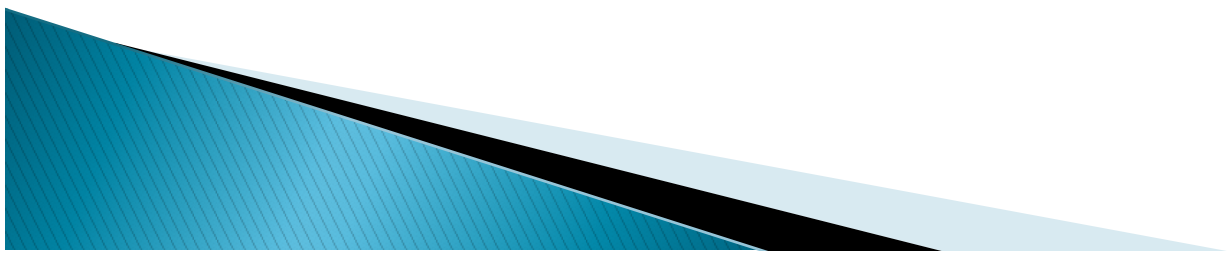
- Dictionaries can thus be used if you are modifying a data file to capture specific data content changes or textual (as opposed to computational) conversions; in the case above, we could use a dictionary to translate terms from English into French.
- This can be an important tool to keep in mind when “scrubbing” data.



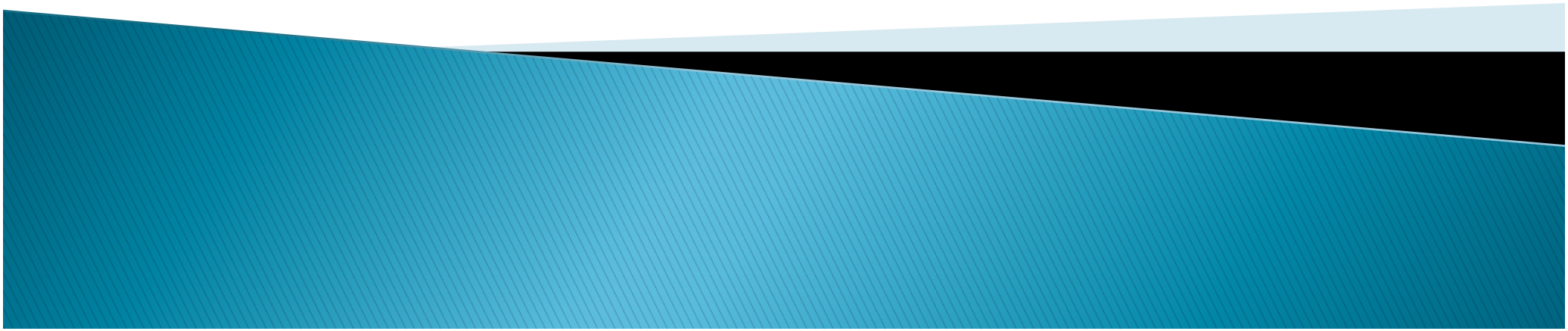
Dictionaries

Python documentation – introduction to dictionaries:

http://docs.python.org/3.1/tutorial/datastructures.html#dictionaries

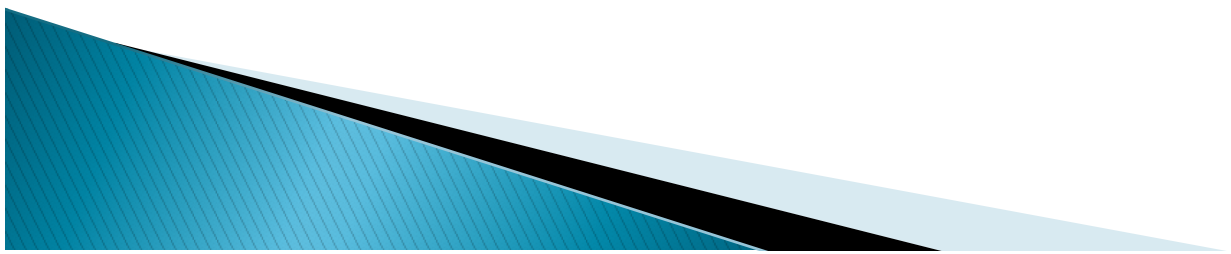


Working with Files



Reading and Writing for Scrubbing Data

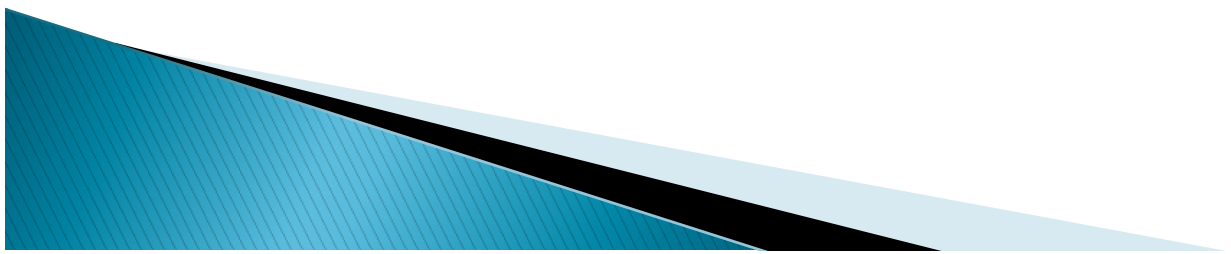
- In order to “scrub” data, one approach is to open two files – one each for reading and writing.
- Your program could then read the data line by line from the first program … and write it to the output file line by line to the second file.



Processing Text Files

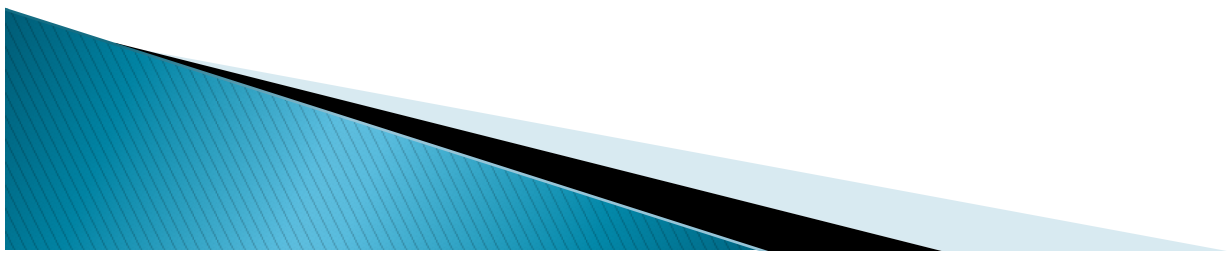
Thus – there are three steps:

1. Open both files
   - Open the source file in “read” mode
   - Open the target file in “write” mode (or “append” mode)
2. Process the file(s)
   - Read the next/appropriate line in the source file
   - If appropriate, write the next modified line of data to the target file
3. Close both files



Python “open file” modes

- The three file modes for opening files in python include:
- ‘r’ – open for reading only
- ‘w’ – open for writing to a new file. *(Note: if a file already exists with this name … it will be deleted.)*
- ‘a’ – open for appending data to the end of an already existing file. *(Note: if a file does not exist with this name … it will be created.)*



The structure of a program to open and read a text file for discussion:

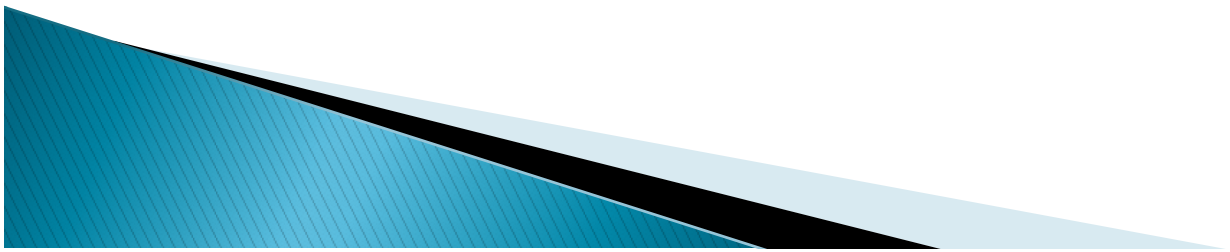
```
def main():
   file_name    = input('Enter the name of the file: ')
   T_file = open(file_name,'r')    # open the file in read mode
   count = 1

    for line in T_file: # loop through the file using
                        # python's "for" loop for this purpose
              text_line = line.rstrip('\n')
         print(str(count)+'. '+ text_line)
         count +=1
    T_file.close()     # close the file
main()
```

Working with numeric data

- Text files that contain statistics (such as statistics from the web or an Excel spreadsheet) would also be read … line by line … but the values would then be cast as integers or as floats; for example:

```
…
 for line in priceFile:
     price = float(line)
…
```



The structure of a program to read, modify and write a text file for discussion:

```
def main():
    file_name1   = "infile.txt"           # filename for the source file
    file_name2   = "outfile.txt"              # filename for the target file
    source_file =  open(file_name1,'r')   # open the source file in "read" mode
    target_file  = open(file_name2,'w')   # open the target file in "write" mode

    for line in source_file:              # use python's "for" loop to read the
                                          # file one line at a time
        text_line = line.rstrip('\n')     # remove the right-most character
        target_file.write(text_line.upper()+'\n') # render the text in capital letters
                                                  # and write it to the output file

     source_file.close()  # close the source file
     target_file.close()  # close the target file

main()
```

It is also a good idea to "catch" File I/O errors with try/except:

```
def main():
    # Get the name of a file.
    filename = input('Enter a filename: ')

    try:
        # Open the file.
        f = open(filename, 'r')

    except IOError:
        print("An error occurred trying to read the file %s " % (filename))

    except Exception as e:
          print("An error occurred. ")
          print(e)

    else:
         // process the data here
main()
```

Python

A final note: We will use Python throughout this semester: to modify data files; build data visualizations; manipulate data in SQLite and other projects.

There are many resources available to help you with Python. Please keep me and the T.A. posted if we can be of any help! I will also post further readings on the course home pages.

- *Deena*