Reading and writing files
Practical Computing for Biologists

Chapter 10 (2nd half)

Peter Brooks
Jan 13 2012

Chapter 10 topics
– illustrated by transforming a text file into a Google Earth .kml file

```
Dive  Date  Lat   Lon   Depth Notes
Tiburon 596 19-Jul-03   36 36.12 N  122 22.48 W 1190  holotype
JSL II 1411 16-Sep-86   39 56.4 N   70 14.3 W   518   paratype
JSL II 930  18-Aug-84   40 05.03 N  69 03.01 W  686   Youngbluth (1989)
Ventana 1575      11-Mar-99   36 42.24 N  122 02.52 W 767
Ventana 1777      16-Jun-00   36 42.60 N  122 02.70 W 934
Ventana 2243      9-Sep-02    36 42.48 N  122 03.84 W 1001
Tiburon 515 24-Nov-02   36 42.00 N  122 01.98 W 1156
Tiburon 531 13-Mar-03   24 19.02 N  109 12.18 W 1144
Tiburon 547 31-Mar-03   24 14.04 N  109 40.02 W 1126
JSL II 3457 26-Sep-03   40 17.77 N  68 06.68 W  862   Francesc Pages (pers.comm)
```

Flat text file
rows and columns (fields)

ML file
Markup Language

Each row parsed between series of hierarchical « tags ».

XML
A lingua franca usable by many programs.
Goog Earth:
Keyhole ML - kml

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://earth.google.com/kml/2.2">
<Document>

<Placemark>
 <name>Tiburon 596</name>
 <description>Tiburon 596    19-Jul-03   36 36.12 N  122 22.48 W 1190  holotype</description>
 <Point>
  <altitudeMode>absolute</altitudeMode>
  <coordinates>-122.374667, 36.602000, -1190</coordinates>
 </Point>
</Placemark>
<Placemark>
 <name>JSL II 1411</name>
 <description>JSL II 1411    16-Sep-86   39 56.4 N   70 14.3 W   518   paratype</description>
 <Point>
  <altitudeMode>absolute</altitudeMode>
  <coordinates>-70.238333, 39.940000, -518</coordinates>
 </Point>
</Placemark>
<Placemark>

(…)

</Placemark>

</Document>
</kml>
```

Chapter 10 topics

– illustrated by transforming a text file into a Google Earth .kml file

- First part Ch 10 Jan. 6

  file parsing strategies
  open(FileName, 'r') and read lines with a for loop
  .strip('\n') remove trailing characters
  .split() parse line elements into lists [….] and access list elements
  open(NewFile, 'w') and OutFile.write()

- Last part Ch 10 Jan. 13

  Reg expr search using re.search()
  .group() to access re.search() results
  create custom functions with def my_function:
  generate XML and KML files
  multiple line strings bounded by triple quotes
  If time: raw strings; re.sub for search and replace

Review previous

Open latlon_3.py

Files – 2 variables
1. file name - “string.txt”
InFileName = 'Marrus_claudanielis.txt'
2. file “handle” points to file
InFile = open(InFileName, 'r')
== == = = =
Counter variable, header
For loop to read each line
Strip line returns.
Split on \t – make list of fields.
Format output and assign output to a variable.
== === =
Create a string for output file name.
Define the output file handle for writing.
Write output.
    to screen
    to file – add \n
Close all files.

Run program to see output.

```
#!/usr/bin/env python
(...)      1      2

LineNumber = 0

# Open the output file for writing -Do this *before* the loop, not inside it
OutFileName=InFileName + ".kml"

OutFile=open(OutFileName,'w') # You can append instead with 'a'

# Loop through each line in the file
for Line in InFile:
    # Skip the header, line # 0
    if LineNumber > 0:
        # Remove the line ending characters
        Line=Line.strip('\n')
        ElementList=Line.split('\t')

        # Use the % operator to generate a string
        # We can use this for output both to the screen and to a file
        OutputString = "Depth: %s\tLat: %s\t Lon:%s" % \
          (ElementList[4], ElementList[2], ElementList[3])

        # Can still print to the screen then write to a file
        print OutputString

        # Unlike print statements, .write needs a linefeed
        OutFile.write(OutputString+"\n")

    # Index the counter used to keep track of line numbers
    LineNumber = LineNumber + 1

# After the loop is completed, close the files
InFile.close()
OutFile.close()
```

Parsing with Regular Expressions in Python
Convert Lat/Long fr degrees-minutes to decimal degrees

- Ch 2 and 3:   Regex in text editor

- Python: Regex commands are in module “re”

  ```
  import re          # import command – follows shebang early in code
  In interactive mode, >>> dir(re) shows available re tools
  ```

- Result = re.search(regexPattern, targetString)

- Extract latitude elements from string using regex

  ```
  See latlon_3.py output
  “39 56.4 N”            example of ElementList[2]
  Regex:  1 or more digits, space, 1 or more digits or decimal pts,
                                        space, a word character ( i.e. a letter)
      \d+ [\d\.]+ \w     But want each as an independent element:
    (\d+)   ([\d\.]+)   (\w)   => only elements in (…...) can be retrieved
  Code lines for program:
      patternLatLon = '(\d+) ([\d\.]+) (\w)'   # the pattern itself is a quoted string
      Result = re.search(patternLatLon, ElementList[2])
  ```

re.search output: an object containing the results of the pattern search and methods to work with results

- Result = re.search(regexPattern, targetString)

  The “Match.Object” Result has methods invoked by the dot operator.

- Result.group()

  Result.group(0) is entire string
  DegreeString = Result.group(1)   # this is equivalent to \1 in Text wrangler
  MinuteString = Result.group(2)
  Compass = Result.group (3)

- These values will be used in a custom function that converts to decimal degrees.

- Other useful methods are available for the search output:

  MinPosition = Result.start(2)   # the index of the start of match of 2nd group
  DegMinEnds = Result.end(1,2)   # one past the end of match of groups 1 and 2

- Any alternate method to get degrees, minutes and compass letter?

  (without using re)

Defining custom functions

- Need to convert both latitude and longitude to decimal degrees.

  Easy to copy code for latitude, and modify to treat longitude.

- Function calls (subroutines) enhance readability, decrease chances of coding errors, and facilitate debugging.

- DRY – Don't Repeat Yourself

  Avoid code containing repetitive lines or blocks of lines.

  Do not copy / paste code to do something similar.

  When corrections or modifications are needed, they are done only in one place.

- Built-in and custom functions are used in the same way.

  Function float(X) operates on a value of a parameter; returns decimal value.

  A custom function processes parameter(s) and returns result(s).

  The function is placed before the main block of code and is called from the main block or from within other functions.

- def decimalat(DegString):

  This line gives the name of the function and parameters.

  Indented lines following the colon contain the “definition” of the function.

- Functions typically end with a “return” of values, but not always.

  Always leave the call of the function with a return statement.

Degree-minutes to Decimal Degrees Function

- Function takes one parameter: DegString
- Strings from search groups are converted to decimal numbers.
- Good habit: do not assume all records are uniform; as group(3) is a string, can use string methods – upper().

```
def decimalat(DegString):  # The call can use other variable names.
        # This function requires that the re module is loaded
        # Take a string in the format "34 56.78 N" and return decimal deg.
        SearchStr= '(\d+) ([\d\.]+) (\w)'
        Result = re.search(SearchStr, DegString)

        # Get the captured character groups, as defined by the parenth.
        #   in the regular expression, convert the numbers to floats, and
        #   assign them to variables with meaningful names
        Degrees = float(Result.group(1))
        Minutes = float(Result.group(2))
        Compass = Result.group(3).upper() # make sure it is capital too

        # Calculate the decimal degrees
        DecimalDegree = Degrees + Minutes/60

        # If the compass direction indicates the coordinate is South or
        #      West, make the sign of the coordinate negative.

        if Compass == 'S' or Compass == 'W':
                DecimalDegree = -DecimalDegree

        return DecimalDegree     # The output sent back to line of call.
# End of the function definition
```

Calling a function from the main block of code

1. Simple, descriptive names.
2. New variable names get the value of the function's “return”. Names of sent variables are different from function's parameter name.
3. Use of backstroke character to continue long lines. Gives good readability.
4. WriteOutFile is a boolean, set as True or False earlier in the code.

from latlon_4.py

```
for Line in InFile:
    if LineNumber > 0:
        # print line  # uncomment for debugging
        Line=Line.strip('\n')
        ElementList = Line.split('\t')
        # Returns a list in this format:
        # ['Tiburon 596', '19-Jul-03', .......]
        # print "ElementList:", ElementList  # uncomment for debug

  1     Dive    = ElementList[0]
        Date     = ElementList[1]
        Depth   = ElementList[4]
        Comment = ElementList[5]

  2     LatDegrees = decimalat(ElementList[2])
        LonDegrees = decimalat(ElementList[3])
        # Create string to 5 decimal places, padded to 10 total char.
        # (using line continuation character \)
  3     OutString = "%s\t%4s\t%10.5f\t%10.5f\t%9s\t%s" % \
              (Dive,Depth,LatDegrees,LonDegrees,Date,Comment)
        print OutString
  4     if WriteOutFile:
            OutFile.write(OutString + '\n') # remember the line feed!
    LineNumber += 1 # this is outside the if, but inside the for loop
# Close the files
```

Converting flat text to XML format

- ML: Markup Language

  Each data element annotated with opening and closing tags.

  ```
  <element_name>Element Content</element name>
  <sample>
      <species>Marrus claudanielis</species>
      <depth>-Si8</depth>
      <location>-70.238, 39.940</location>
  </sample>
  ```

  Sets of tags are always nested.

- ML file headers and footers – also use opening and closing tags

  (Use triple quotes for multiline strings or comments.)

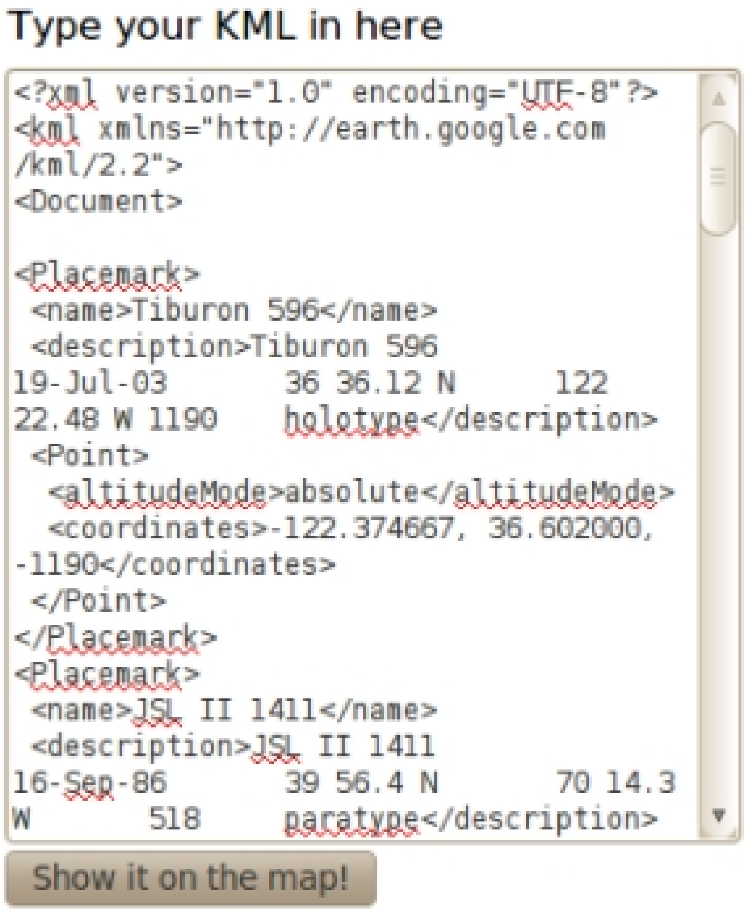
  ```
  HeadString=' ' '<?xml version=\"1.0\" encoding=\"UTF-8\"?> (no closing tag)
  <kml xmlns=\"http://earth.google.com/kml/2.2\">
  <Document>' ' '
  Footer: </Document>    </kml>      #  Note that tags are nested.
  ```

- Placemark string to build KML records.

  See latlon_5.py for multiline example.

  To preserve data not needed by program (Google Earth), include a Description field that has the entire original data line.

A .kml file can be visualized with Google Earth or Google Map

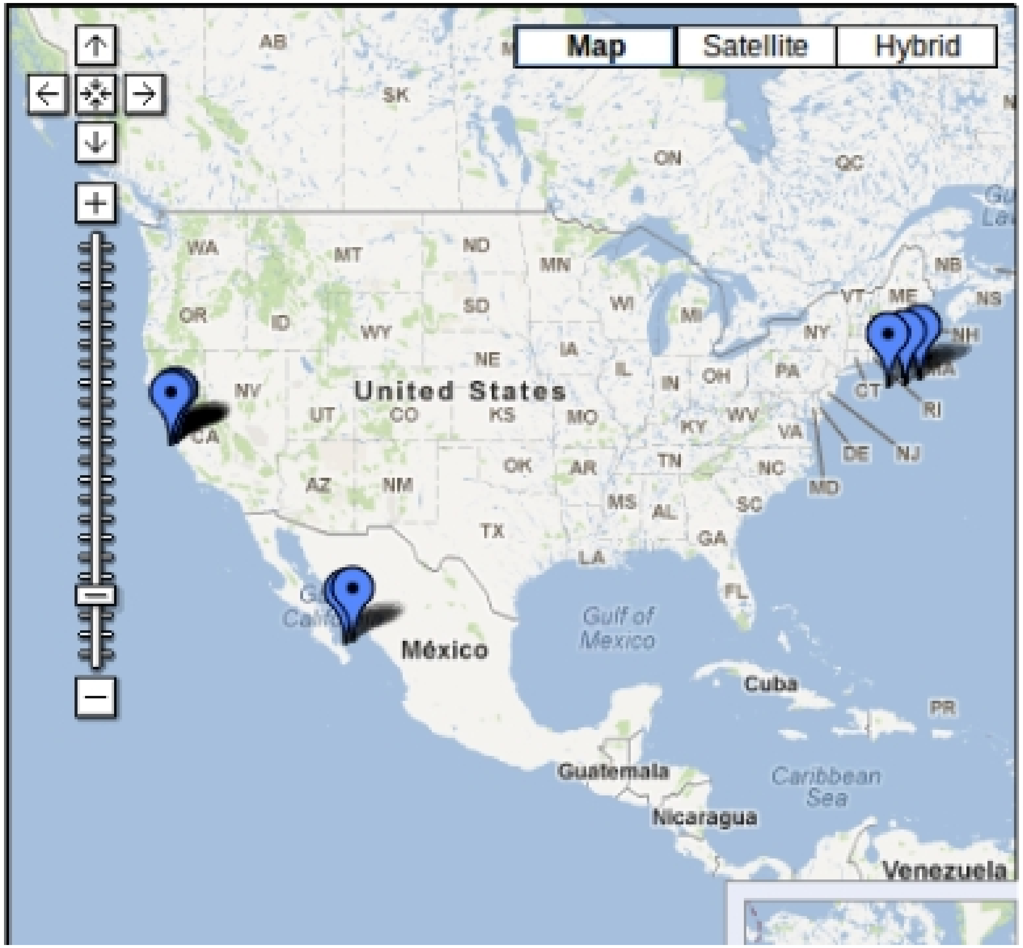


Type your KML in here

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://earth.google.com
/kml/2.2">
<Document>

<Placemark>
 <name>Tiburon 596</name>
 <description>Tiburon 596
19-Jul-03        36 36.12 N       122
22.48 W 1190     holotype</description>
 <Point>
  <altitudeMode>absolute</altitudeMode>
  <coordinates>-122.374667, 36.602000,
-1190</coordinates>
 </Point>
</Placemark>
<Placemark>
 <name>JSL II 1411</name>
 <description>JSL II 1411
16-Sep-86        39 56.4 N        70 14.3
W       518      paratype</description>
```

Show it on the map!



http://display-kml.appspot.com/

Raw strings

```
s=" c: \ \ "
        Python will escape the backslash
                resulting string s will contain only one backstroke
                value  is  "c: \"
raw string      –     precede string with letter r
        s=r" c: \ \ "
                suppresses Python's character escaping,
                value is "c : \ \ "
```

Use raw string even if uncertain it is necessary.
===== = = = = = = =

Substitutions – Search and Replace

re.sub()

```
>>> import re
>>> OrigString = "Haeckel, Ernst"   # The string you wish to search
>>> SubFind = r"( \ w+) , ( \ w+)"     # The regular expression to search for
>>> Result = re. search (SubFind, OrigString)      # Perform the search, store the results
>>> print Result.groups()   #  See  what you found
('Haeckel', 'Ernst')               #  The two captured substrings
>>> SubReplace = r" \ 2 \ t \ 1"        #  Set up a string for replacement using raw string style
>>> NewString = re.sub(SubFind, SubReplace, OrigString)      #  Format for re.sub( )
>>> print NewString    #  See the substitUted string
'Ernst tab Haeckel'
```