NICHOLAS SCHOOL OF THE ENVIRONMENT AND EARTH SCIENCES

DUKE UNIVERSITY

Introduction to **Scripting**: Writing Python Scripts

ENV 859

Geospatial Data Analytics

Learning Objectives

- **The *process* of writing a Python script**
  - Objectives and approaches
  - Best practices

- **More practice on…**
  - Variables & data types
  - File objects
  - Iteration *(for… & while…* loops*)*
  - Conditional processing *(if…else…)*
  - Handling script errors
  - User input

The Zen of Python, by Tim Peters

Beautiful is better than ugly.

Explicit is better than implicit.

Simple is better than complex.

Complex is better than complicated.

Flat is better than nested.

Sparse is better than dense.

Readability counts.

Special cases aren't special enough to break the rules,

Although practicality beats purity.

Errors should never pass silently,

Unless explicitly silenced.

In the face of ambiguity, refuse the temptation to guess.

There should be one—and preferably only one—obvious way to do it.

Although that way may not be obvious at first unless you're Dutch.

Now is better than never.

Although never is often better than *right* now.

If the implementation is hard to explain, it's a bad idea.

If the implementation is easy to explain, it may be a good idea.

Namespaces are one honking great idea—let's do more of those!

http://proceedings.esri.com/library/userconf/proc04/docs/pap1027.pdf

The Task

Your research team just caught wind that you know Python!

They have some ARGOS tracking data - a text file in a marginally human readable format and with a lot of “noise”.

They want you to build a tool whereby a user can enter a date and retrieve the location(s) at which the turtle was observed.

The Task

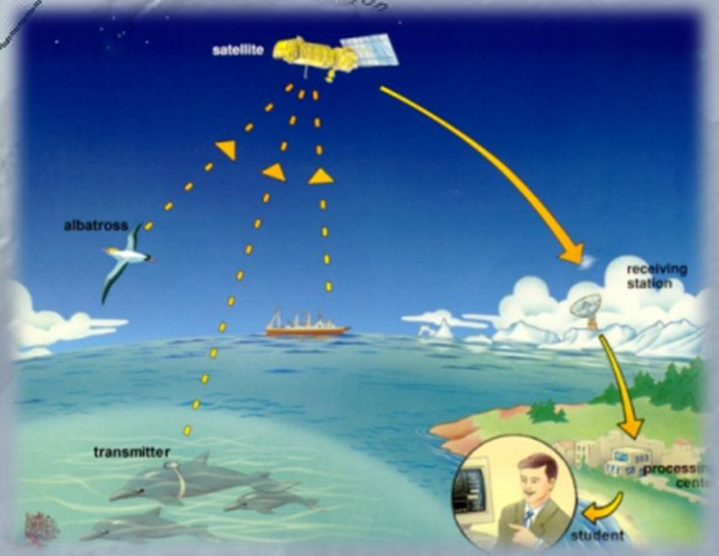
Sara.txt

```
# lc    location class, see http://www.nacls.com/html/argos/manual/html/chap2/chap2_3.htm#2.3.5
# iq    quality index, see http://www.nacls.com/html/argos/manual/html/chap4/chap4_4_9.htm#4.4.9.3
# lat1  solution1 latitude
# lon1  solution1 longitude
# lat2  solution2 latitude
# lon2  solution2 longitude
# nb_mes    number of messages received
# big_nb_mes    number of messages received by satellite with signal strength greater than -120 decibels
# best_level    strongest signal strength received
# pass_duration time elapsed between first and last messages being received by satellite
# nopc  number of successful plausibility checks (0 to 4)
# calcul_freq   calculated transmit frequency
# altitude  altitude used in location calculation
uid tag_id  utc lc  iq  lat1    lon1    lat2    lon2    nb_mes  big_nb_mes  best_level  pass_duration   nopc    calcul_freq altitu
20616   29051   7/3/2003 9:13   3   66  33.898  -77.958 27.369  -46.309 6   0   -126    529 3   401 651134.7    0
21892   29051   7/3/2003 9:23   B   0   33.887  -77.95  43.983  -128.418    2   0   -132    96  1   401 651133.1    0
20618   29051   7/3/2003 10:31  A   6   33.884  -77.946 22.875  -25.276 3   0   -132    320 3   401 651169.3    0
20619   29051   7/3/2003 10:49  A   7   33.927  -78.008 37.291  -94.435 3   0   -125    260 2   401 651166.8    0
20620   29051   7/3/2003 12:10  2   67  33.901  -77.909 31.005  -64.614 4   0   -124    612 3   401 651181.6    0
20621   29051   7/3/2003 12:11  B   0   33.575  -77.691 32.741  -73.276 2   0   -126    408 2   401 651134.7    0
20624   29051   7/3/2003 14:45  B   0   33.84   -77.807 26.446  -41.968 2   0   -134    151 2   401 651134.7    0
20627   29051   7/3/2003 16:28  B   0   33.794  -77.513 36.337  -89.9   2   0   -130    281 2   401 651134.7    0
20629   29051   7/3/2003 17:02  A   8   33.95   -77.804 36.8    -91.697 3   0   -121    186 3   401 651176.7    0
20630   29051   7/3/2003 17:29  B   0   33.965  -77.984 42.107  -38.299 2   0   -127    237 2   401 651134.7    0
20631   29051   7/3/2003 19:07  0   58  33.989  -77.698 32.239  -86.421 4   0   -125    514 3   401 651179.0    0
21893   29051   7/3/2003 22:11  B   0   34.028  -77.74  30.15   -96.516 2   0   -124    191 2   401 651179.0    0
20639   29051   7/4/2003 2:32   B   0   34.117  -77.735 36.995  -63.539 2   0   -127    143 2   401 651179.0    0
20640   29051   7/4/2003 3:42   B   0   34.136  -77.725 31.25   -91.766 2   0   -126    53  2   401 651179.0    0
```

Exercise: Process ARGOS Data

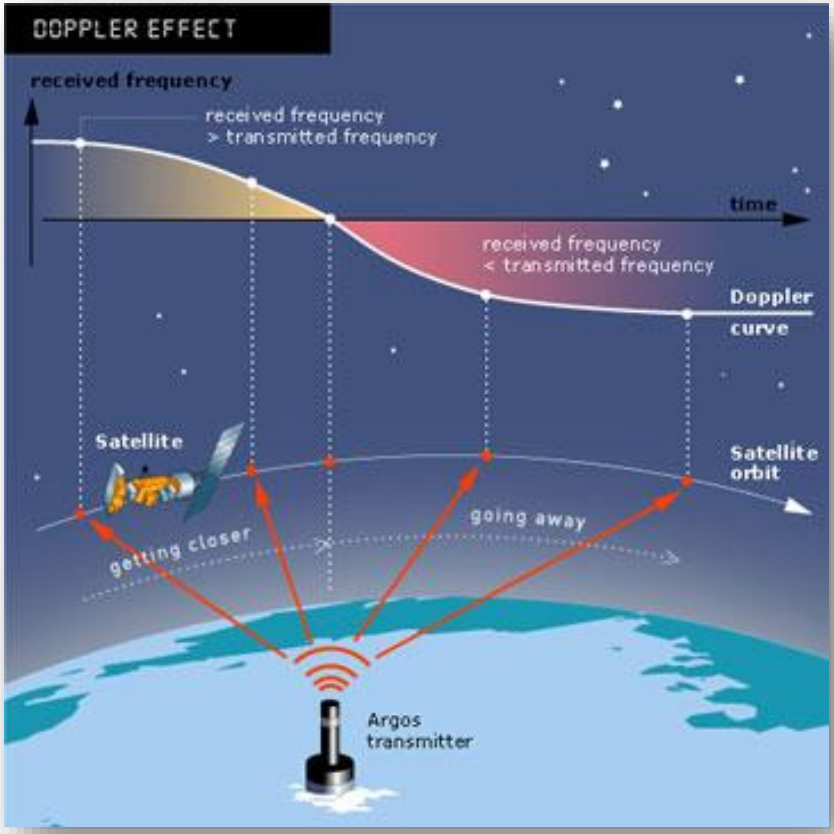
ARGOS data: "Sara"

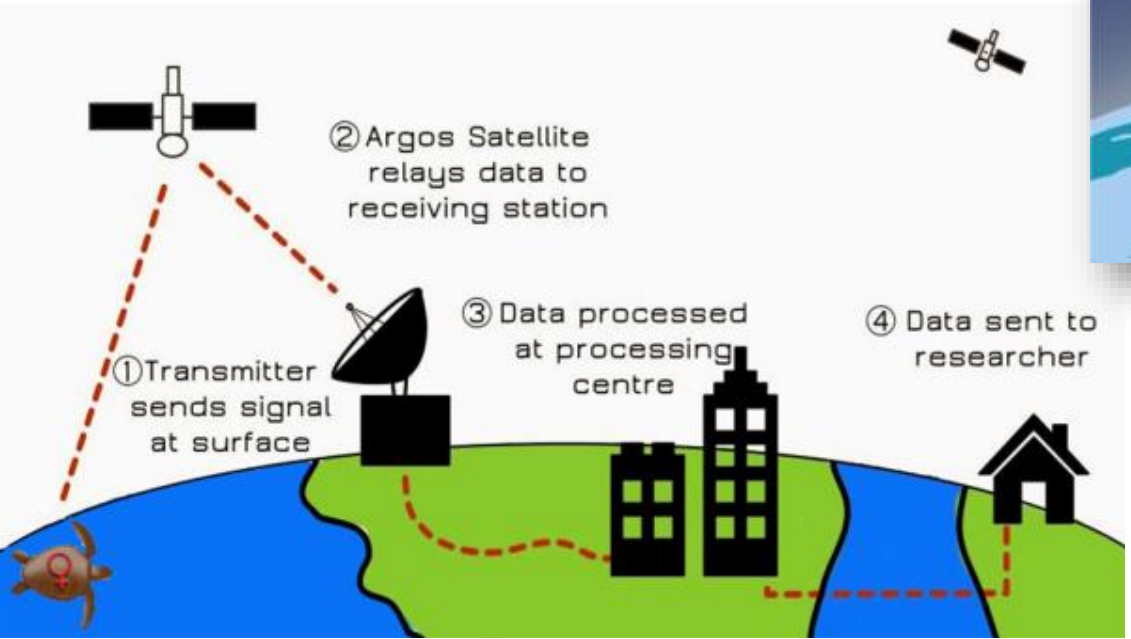
- Allow user to pick a date and display information on any observations recorded that day



How ARGOS works…

https://conserveturtles.org/sea-turtle-tracking-works





Exercise: Process ARGOS Data

ARGOS data: "Sara"

Sara.txt

```
# lc    location class, see http://www.nacls.com/html/argos/manual/html/chap2/chap2_3.htm#2.3.5
# iq    quality index, see http://www.nacls.com/html/argos/manual/html/chap4/chap4_4_9.htm#4.4.9.3
# lat1  solution1 latitude
# lon1  solution1 longitude
# lat2  solution2 latitude
# lon2  solution2 longitude
# nb_mes    number of messages received
# big_nb_mes    number of messages received by sate
# best_level    strongest signal strength received
# pass_duration time elapsed between first and last messages being received by satellite
# nopc  number of successful plausibility checks (0 to 4)
# calcul_freq   calculated transmit frequency
# altitude  altitude used in location calculation
uid tag_id  utc lc  iq  lat1    lon1    lat2    lon2    nb_mes  big_nb_mes  best_level  pass_duration   nopc    calcul_freq altitu
20616   29051   7/3/2003 9:13   3   66  33.898  -77.958 27.369  -46.309 6   0   -126    529 3   401 651134.7    0
21892   29051   7/3/2003 9:23   B   0   33.887  -77.95  43.983  -128.418    2   0   -132    96  1   401 651133.1    0
20618   29051   7/3/2003 10:31  A   6   33.884  -77.946 22.875  -25.276 3   0   -132    320 3   401 651169.3    0
20619   29051   7/3/2003 10:49  A   7   33.927  -78.008 37.291  -94.435 3   0   -125    260 2   401 651166.8    0
20620   29051   7/3/2003 12:10  2   67  33.901  -77.909 31.005  -64.614 4   0   -124    612 3   401 651181.6    0
20621   29051   7/3/2003 12:11  B   0   33.575  -77.691 32.741  -73.276 2   0   -126    408 2   401 651134.7    0
20624   29051   7/3/2003 14:45  B   0   33.84   -77.807 26.446  -41.968 2   0   -134    151 2   401 651134.7    0
20627   29051   7/3/2003 16:28  B   0   33.794  -77.513 36.337  -89.9   2   0   -130    281 2   401 651134.7    0
20629   29051   7/3/2003 17:02  A   8   33.95   -77.804 36.8    -91.697 3   0   -121    186 3   401 651176.7    0
20630   29051   7/3/2003 17:29  B   0   33.965  -77.984 42.107  -38.299 2   0   -127    237 2   401 651134.7    0
20631   29051   7/3/2003 19:07  0   58  33.989  -77.698 32.239  -86.421 4   0   -125    514 3   401 651179.0    0
21893   29051   7/3/2003 22:11  B   0   34.028  -77.74  30.15   -96.516 2   0   -124    191 2   401 651179.0    0
20639   29051   7/4/2003 2:32   B   0   34.117  -77.735 36.995  -63.539 2   0   -127    143 2   401 651179.0    0
20640   29051   7/4/2003 3:42   B   0   34.136  -77.725
```

7/3/2003

Enter a date [M/D/YYYY] (Press 'Enter' to confirm or 'Escape' to cancel)

On 7/3/2003, Sara was seen at 33.898,-77.958
On 7/3/2003, Sara was seen at 33.901,-77.909

Step 1: *Pseudocode*

ARGOS data: "Sara"

1. Open ARGOS data file
2. Read and parse each line
   a. Skip comment lines
   b. Skip records below a quality threshold (*qc* <> 1, 2, or 3)
   c. Add obs. *date* to a date dictionary, keyed by *UID*
   d. Add obs. *lat/long* to a location dictionary, keyed by *UID*
3. Allow user to specify date
   a. Inform if date is invalid
4. Identify keys in date dictionary matching user supplied date
5. Identify values in location dictionary with keys found above
6. Print information to screen

Plan of attack: *start simple*

Sequence	Task
1.	Parse a single line of tracking data into variables
2.	Read a single line of tracking data from the file into Python (and then parse into variables)
3.	Read in all lines of tracking data from a file into Python (and then parse into variables)
4.	While reading in all lines of tracking data, add variables into dictionaries of values
5.	Add conditional statements so only valid values are added to the dictionaries.
6.	From our dictionaries, extract data for a selected date.
7.	All the user to specify the date used to select data from our dictionary
8.	Add code to handle if the user enters an improper date

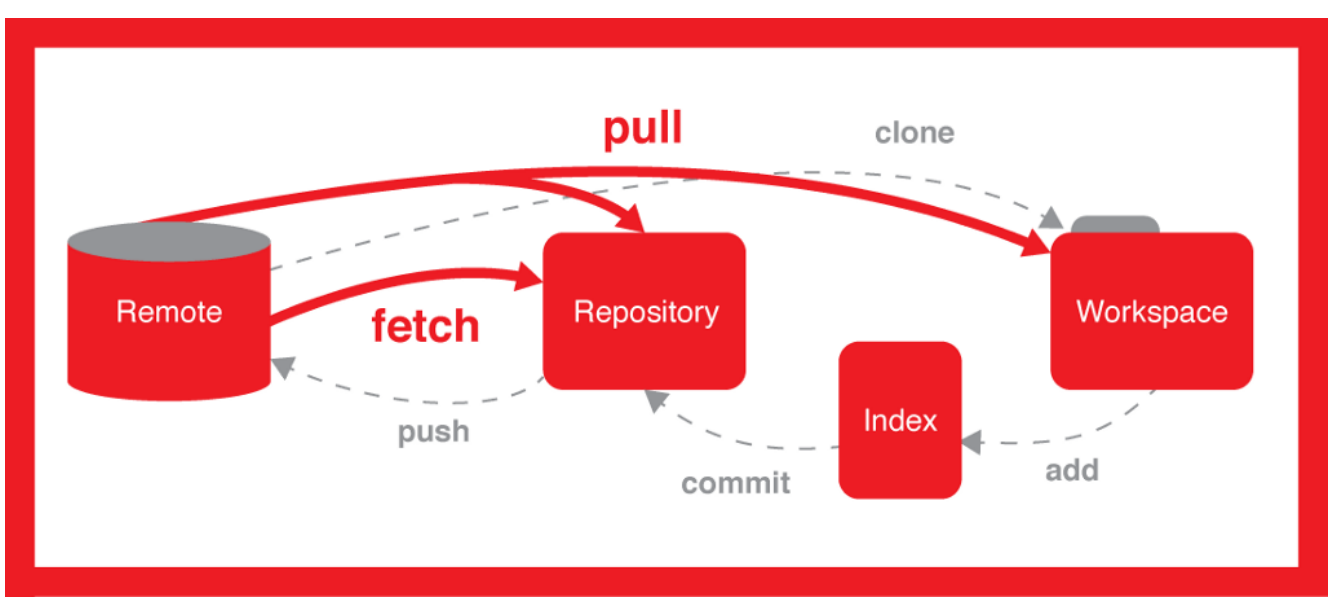
What’s next?

Coding platform: VS Code

Versioning software: Git/GitHub

Practice writing code!

Intro to Git/GitHub



Parsing strings into variables

- Task1.py -Parse a line of tracking data

Task1.py

```
1  # Task1.py
2  #
3  # Description: Parses a line of ARGOS tracking data
4  #
5  # Created by: John Fay (jpfay@duke.edu)
6  # Created on: Oct 2011
7
8  # Copy and paste a line of data as the startLine variable value
9  lineString = "20616 29051   7/3/2003 9:13   3   66  33.898  -77.958 27.369  -46.309 6   0   -126
10
11 # Use the split command to parse the items in lineString into a list object
12 lineData = lineString.split("\t")
13
14 # Assign variables to specfic items in the list
15 recordID = lineData[0]              # ARGOS tracking record ID
16 obsDateTime = lineData[2]           # Observation date and time (combined)
17 obsDate = obsDateTime.split()[0]    # Observation date - first item in obsDateTime list object
18 obsTime = obsDateTime.split()[1]    # Observation time - second item in obsDateTime list object
19 obsLC = lineData[3]                 # Observation Location Class
20 obsLat = lineData[5]                # Observation Latitude
21 obsLon = lineData[6]                # Observation Longitude
22
23 # Print information to the user
24 print "According to record " + recordID,
25 print "Sara was seen at " + str(obsLat) + " d LAT; " + str(obsLat) + " d LON"
```

Python *file objects*

(for Task2 which reads data from an ARGOS data file)

http://docs.python.org/release/2.6.5/tutorial/inputoutput.html#reading-and-writing-files

7.2. Reading and Writing Files

`open()` returns a file object, and is most commonly used with two arguments: `open(filename, mode)`.

```
>>> f = open('/tmp/workfile', 'w')
>>> print f
<open file '/tmp/workfile', mode 'w' at 80a0960>
```

The first argument is a string containing the filename. The second argument is another string containing a few characters describing the way in which the file will be used. *mode* can be `'r'` when the file will only be read, `'w'` for only writing (an existing file with the same name will be erased), and `'a'` opens the file for appending; any data written to the file is automatically added to the end. `'r+'` opens the file for both reading and writing. The *mode* argument is optional; `'r'` will be assumed if it's omitted.

On Windows, `'b'` appended to the mode opens the file in binary mode, so there are also modes like `'rb'`, `'wb'`, and `'r+b'`. Python on Windows makes a distinction between text and binary files; the end-of-line characters in text files are automatically altered slightly when data is read or written. This behind-the-scenes modification to file data is fine for ASCII text files, but it'll corrupt binary data like that in `JPEG` or `EXE` files. Be very careful to use binary mode when reading and writing such files. On Unix, it doesn't hurt to append a `'b'` to the mode, so you can use it platform-independently for all binary files.

7.2.1. Methods of File Objects

The rest of the examples in this section will assume that a file object called `f` has already been created.

Python *file objects*

Python "file object"

file object

file name

- open a file as read-only object `>>> f = open(fn, 'r')`
- open a file for writing (erases if exists) `>>> f = open(fn, 'w')`
- open a file for appending lines to it `>>> f = open(fn, 'a')`

- read the first line from a file object `>>> print f.readline()`
  moves the file pointer to the next line
- read *all* lines from the text file into a list object `>>> data = f.readlines()`

- write to the file `>>> f.write("Hi!\n")`

- close the file `>>> f.close()`

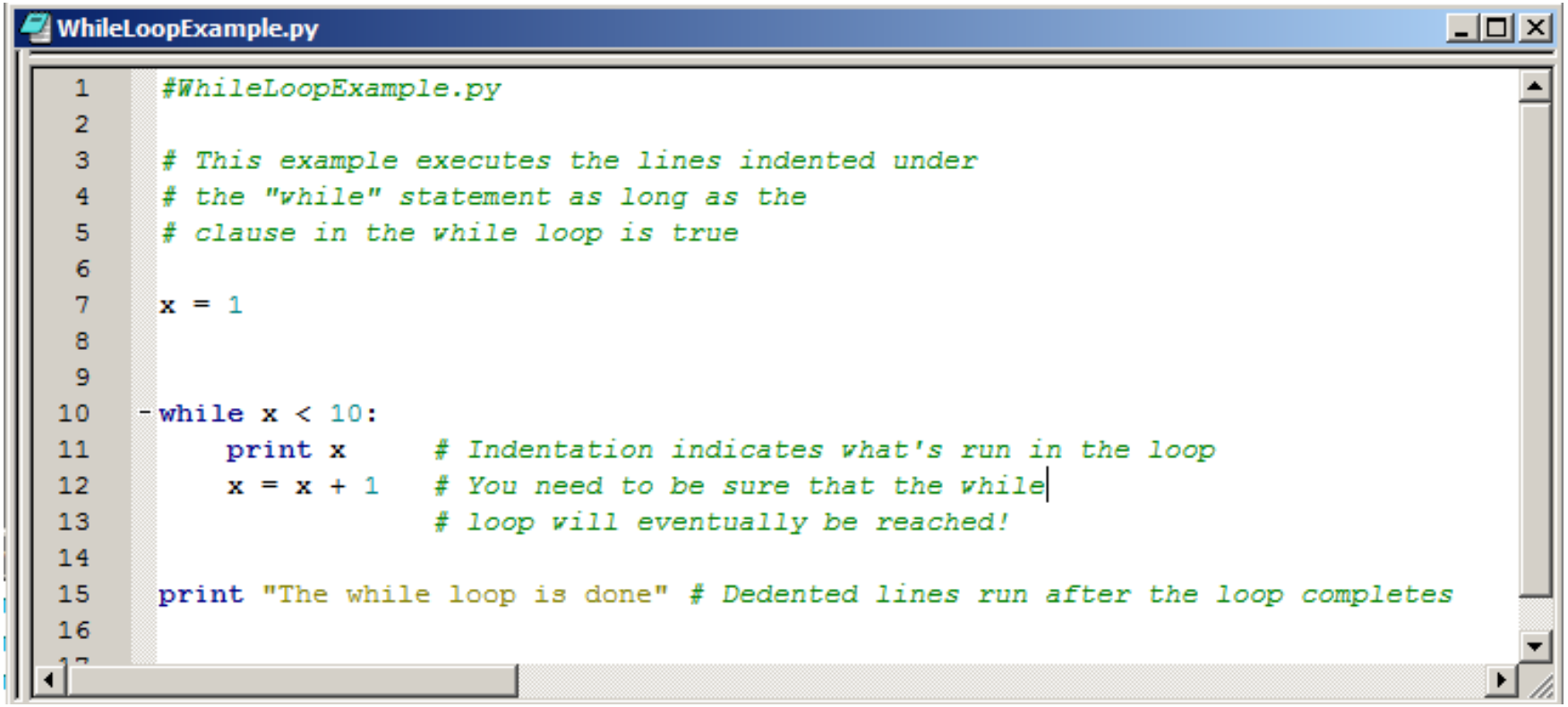
Task 2: Read a line from ARGOS file

- Task2.py – *Reads in first line of data from a text file (rather than having to paste it in the script itself)*

Task2.py

```
1   # Task2.py
2   #
3   # Description: Reads in ARGOS data file and parses a line of ARGOS tracking data
4   #
5   # Created by: John Fay (jpfay@duke.edu)
6   # Created on: Oct 2011
7
8   # Create a variable pointing to the file with no header
9   fileName = "S:\\Scripting2\\SaraNoHeader.txt"
10
11  # Open the file as a read-only file object
12  fileObj = open(fileName, 'r')
13
14  # Read the first line from the open file object
15  lineString = fileObj.readline()
16
17  # Close the file object
18  fileObj.close()
19
20  # Use the split command to parse the items in lineString into a list object
21  lineData = lineString.split("\t")
22
23  # Assign variables to specfic items in the list
24  recordID = lineData[0]              # ARGOS tracking record ID
25  obsDateTime = lineData[2]           # Observation date and time (combined)
```

While loops



Indentation *is a key feature of Python*

Task 3: Read all data from ARGOS file

- Task3.py – *Use a while loop to read all lines from the ARGOS file*

Read first line:
lineString has a value

While loop continues as long as *lineString* has a value

Indented lines are run only as part of while loop.

Update the *lineString* value to the next line

Close the file object

Task3.py

```
1  # Task3.py
2  #
3  # Description: Reads in ARGOS data and displays information to the user
4  #
5  # Created by: John Fay (jpfay@duke.edu)
6  # Created on: Oct 2011
7
8  # Create a variable pointing to the file with no header
9  fileName = "S:\\Scripting2\\SaraNoHeader.txt"
10
11 # Open the file as a read-only file object
12 fileObj = open(fileName, 'r')
13
14 # Read the first line from the open file object
15 lineString = fileObj.readline()
16
17 # Use a while loop to read each line, one at a time, until the end of the file is reached
18 while lineString:
19
20     # Use the split command to parse the items in lineString into a list object
21     lineData = lineString.split("\t")
22
23     # Assign variables to specfic items in the list
24     recordID = lineData[0]              # ARGOS tracking record ID
25     obsDateTime = lineData[2]           # Observation date and time (combined)
26     obsDate = obsDateTime.split()[0]    # Observation date - first item in obsDateTime list object
27     obsTime = obsDateTime.split()[1]    # Observation time - second item in obsDateTime list object
28     obsLC = lineData[4]                 # Observation Location Class
29     obsLat = lineData[5]                # Observation Latitude
30     obsLon = lineData[6]                # Observation Longitude
31
32     # Print information to the user
33     print "According to record " + recordID,
34     print "Sara was seen at " + str(obsLat) + " d LAT; " + str(obsLat) + " d LON"
35
36     # Read in the next line
37     lineString = fileObj.readline()
38
39 # Close the file object
40 fileObj.close()
```

For loops

ForLoopExample.py

```
1   #ForLoopExample.py
2
3   # This example executes the lines indented under
4   # the "while" statement as long as the
5   # clause in the while loop is true
6
7   #Create a tuple of days
8   days = ("Su","M","T","W","Th","F","Sa")
9
10  # Loop through each item in the tuple and execute
11  # each line that is indented under the for loop
12  for day in days:
13      print day
14
15  # Dedent lines run after the loop completes
16  print "The week is over"
```

Task 4: Read all data from ARGOS file

- Task4.py – *Use a for loop to process all lines from the ARGOS file*

Reads in all lines, creating a list object (*lineStrings)*

Prints the number of records in the list

Closes the file object

Iterates through each item in the *lineStrings* list.

Task4.py

```
1  # Task4.py
2  #
3  # Description: Reads in ARGOS data using a FOR LOOP
4  #
5  # Created by: John Fay (jpfay@duke.edu)
6  # Created on: Oct 2011
7
8  # Create a variable pointing to the file with no header
9  fileName = "S:\\Scripting2\\SaraNoHeader.txt"
10
11 # Open the file as a read-only file object
12 fileObj = open(fileName, 'r')
13
14 # Read the first line from the open file object
15 lineStrings = fileObj.readlines()
16 print "There are " + str(len(lineStrings)) + " records in the file"
17
18 # Close the file object
19 fileObj.close()
20
21 # Use a for loop to read each line, one at a time, until the list is exhausted
22 for lineString in lineStrings:
23
24     # Use the split command to parse the items in lineString into a list object
25     lineData = lineString.split("\t")
26
27     # Assign variables to specfic items in the list
28     recordID = lineData[0]              # ARGOS tracking record ID
29     obsDateTime = lineData[2]           # Observation date and time (combined)
30     obsDate = obsDateTime.split()[0]    # Observation date - first item in obsDateTime list object
31     obsTime = obsDateTime.split()[1]    # Observation time - second item in obsDateTime list object
32     obsLC = lineData[4]                 # Observation Location Class
33     obsLat = lineData[5]                # Observation Latitude
34     obsLon = lineData[6]                # Observation Longitude
35
36     # Print information to the user
37     print "According to record " + recordID,
38     print "Sara was seen at " + str(obsLat) + " d LAT; " + str(obsLat) + " d LON"
39
40
```

Task 5a: Create a dictionary of observations

- Task5a.py – *Inserts select ARGOS attributes into dictionaries*

Create two dictionaries:
One for date and one for location; these will be empty at first…

Add values to each dictionary within the for loop; set the record value as the *key* and the date/location data as the *values*.

```
14  fileObj = open(fileName, 'r')
15
16  # Read the first line from the open file object
17  lineStrings = fileObj.readlines()
18  print "There are " + str(len(lineStrings)) + " records in the file"
19
20  # Close the file object
21  fileObj.close()
22
23  # Create empty dictionaries
24  dateDict = {}
25  locationDict = {}
26
27  # Use a for loop to read each line, one at a time, until the list is exhausted
28  for lineString in lineStrings:
29
30      # Use the split command to parse the items in lineString into a list object
31      lineData = lineString.split("\t")
32
33      # Assign variables to specfic items in the list
34      recordID = lineData[0]              # ARGOS tracking record ID
35      obsDateTime = lineData[2]           # Observation date and time (combined)
36      obsLC = lineData[4]                 # Observation Location Class
37      obsLat = lineData[5]                # Observation Latitude
38      obsLon = lineData[6]                # Observation Longitude
39
40      # Add values to dictionary
41      dateDict[recordID] = obsDateTime.split()
42      locationDict[recordID] = (obsLat, obsLon)
43
44  # Indicate script is complete
45  print "Finished"
46
```

If…else…statements

IfElseExample.py

```
1  #IfElseExample.py
2
3  # Loops through the days of the week.
4  # If the current day of the week is weekend, display a message
5  # If the day is Wednesday, print a different message
6
7  #Create a tuple of days
8  days = ("Su","M","T","W","Th","F","Sa")
9
10 # Loop through each day in the tuple of days
11 for day in days:
12     # Evaluate the value
13     if day == "Su" or day == "Sa":
14         print day + " is a weekend day"
15     elif day == "W":
16         print day + ' is "hump day"'
17     else:
18         print day + " is just another day..."
19
20 # Dedent lines run after the loop completes
21 print "The week is over"
```

- Use = to set a variable value;
- Use == to evaluate equivalency

Task 5b: Filter which records are used

- Task5b.py – *Inserts selected ARGOS records into dictionaries*

Create a variable to count records that get omitted

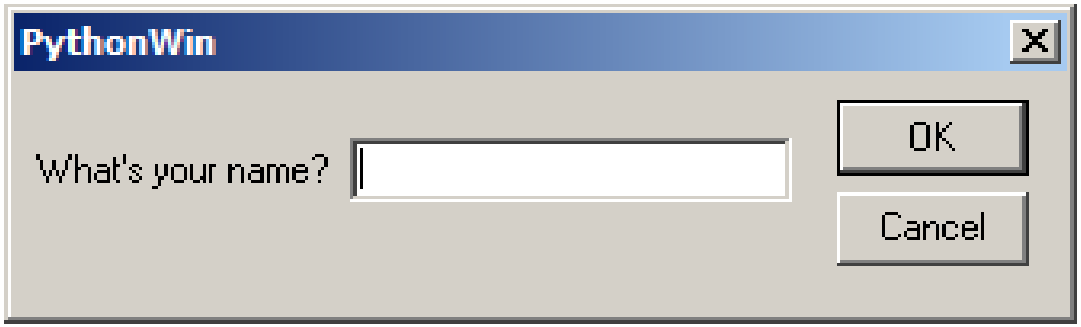
Add value to the dictionaries only if the location class value is 1, 2, or 3

If the record is not added, add to the tally of omitted records

```
19
20  # Close the file object
21  fileObj.close()
22
23  # Create empty dictionaries
24  dateDict = {}
25  locationDict = {}
26  omittedRecordCount = 0
27
28  # Use a for loop to read each line, one at a time, until the list is exhausted
29  for lineString in lineStrings:
30
31      # Use the split command to parse the items in lineString into a list object
32      lineData = lineString.split("\t")
33
34      # Assign variables to specfic items in the list
35      recordID = lineData[0]              # ARGOS tracking record ID
36      obsDateTime = lineData[2]           # Observation date and time (combined)
37      obsLC = lineData[3]                 # Observation Location Class
38      obsLat = lineData[5]                # Observation Latitude
39      obsLon = lineData[6]                # Observation Longitude
40
41      if obsLC in ("1","2","3"):
42          # Add values to dictionary
43          dateDict[recordID] = obsDateTime.split()
44          locationDict[recordID] = (obsLat, obsLon)
45      else:
46          omittedRecordCount = omittedRecordCount + 1
47
48  # Indicate script is complete
49  print str(len(dateDict)) + " records added"
50  print str(omittedRecordCount) + " records omitted"
```

Model inputs: User Input

```
>>> myName = raw_input("What's your name?")
```




```
>>> print "Hello " + myName
Hello John
```

Task 6a: Allow user to select site

Task6a.py

- Use the raw_input() function to get user date

```
53  # Ask the user to enter a start record and and end record
54  userDate = raw_input("Enter date of record (M/D/YYYY):")
55
56  # Create a list of all the dictionary keys (UIDs) with a date matching the user date
57  keyList = []                        #Create an empty list to which we can add keys of matching items
58  for k in dateDict.keys():           #Loop through all the keys in the dateDict
59      v = dateDict[k]                 #Get the value corresponding to the key in the current loop iteration
60      dateValue = v[0]                #The current value is a date, time tuple. Date is the first item in that tuple
61      if dateValue == userDate:       #Check whether the date matches the user date
62          keyList.append(k)           #If it does, then add the key to the key list
63
64  # Now that we have a list of keys, we can loop through them, extract the lat/long values and report them
65  print "At " + userDate + ", Sara the turtle was found at:"
66  for k in keyList:                                   #Loop through the keys identified above
67      userLoc = locationDict[k]                       #Get the lat/long tuple for the current key
68      print " Lat: "+userLoc[0]+"; Lon: "+userLoc[1]  #Print them to the screen...
```

- Create an empty list called keyList…
- Loop through keys in dateDict
  - for each *key*, get the *value*; for each *value*, get the *date*
  - if the *date* matches the *user date*, add the *key* to a *keyList*

Task 6a: Allow user to select site

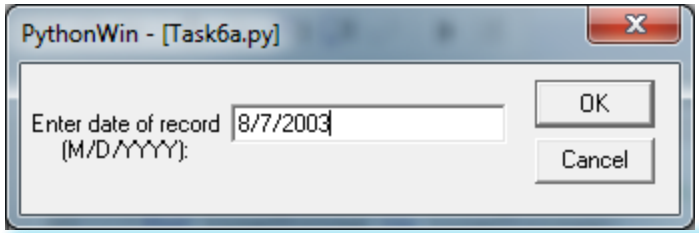
Task6a.py

```
53  # Ask the user to enter a start record and and end record
54  userDate = raw_input("Enter date of record (M/D/YYYY):")
55
56  # Create a list of all the dictionary keys (UIDs) with a date matching the user date
57  keyList = []                    #Create an empty list to which we can add keys of matching items
58  for k in dateDict.keys():       #Loop through all the keys in the dateDict
59      v = dateDict[k]             #Get the value corresponding to the key in the current loop iteration
60      dateValue = v[0]            #The current value is a date, time tuple. Date is the first item in that tuple
61      if dateValue == userDate: #Check whether the date matches the user date
62          keyList.append(k)       #If it does, then add the key to the key list
63
64  # Now that we have a list of keys, we can loop through them, extract the lat/long values and report them
65  print "At " + userDate + ", Sara the turtle was found at:"
66  for k in keyList:                                   #Loop through the keys identified above
67      userLoc = locationDict[k]                       #Get the lat/long tuple for the current key
68      print " Lat: "+userLoc[0]+"; Lon: "+userLoc[1]  #Print them to the screen...
```

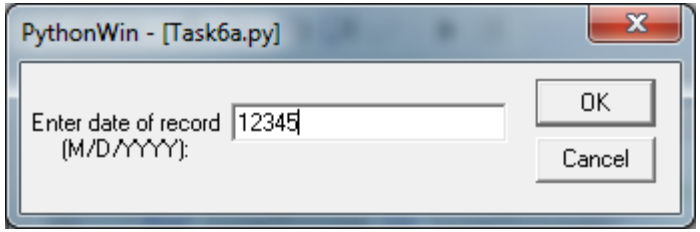
- Loop through the keys in the keyList (i.e. where the date matches)
- Get the corresponding location value from the locationDict
- Print the latitude and longitude values nicely to the screen

Task 6a: Allow user to select site

Task6a.py



```
At 8/7/2003, Sara the turtle was found at:
 Lat: 36.105; Lon: -75.601
 Lat: 36.067; Lon: -75.545
```

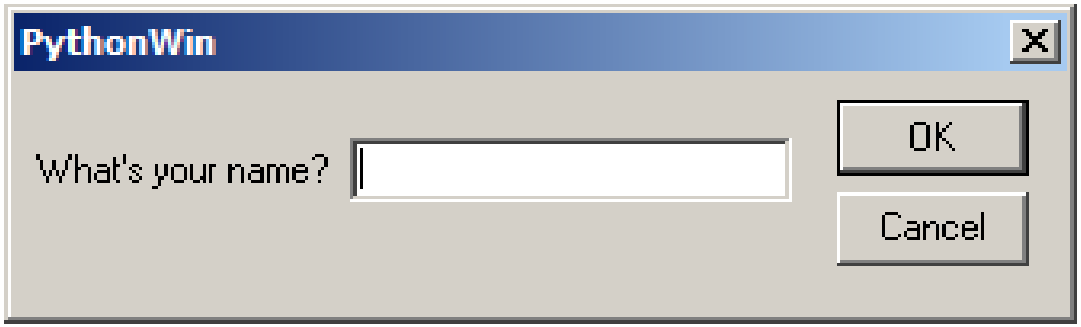


```
At 12345, Sara the turtle was found at:
```

```
At 3:45 on 36.144, Sara the turtle was observed at Lat: 36.144; Lon: -75.442
There are 1124 records in the file
140 records added
984 records omitted
Traceback (most recent call last):
  File "C:\Python26\ArcGIS10.0\Lib\site-packages\Pythonwin\pywin\framework\scriptutils.py", line 322, in RunScript
    debugger.run(codeObject, __main__.__dict__, start_stepping=0)
  File "C:\Python26\ArcGIS10.0\Lib\site-packages\Pythonwin\pywin\debugger\__init__.py", line 60, in run
    _GetCurrentDebugger().run(cmd, globals,locals, start_stepping)
  File "C:\Python26\ArcGIS10.0\Lib\site-packages\Pythonwin\pywin\debugger\debugger.py", line 655, in run
    exec cmd in globals, locals
  File "S:\Scripting2\Task6a.py", line 57, in <module>
    userLoc = locationDict[userRec]
KeyError: '12345'
```

Model inputs: User Input

```
>>> myName = raw_input("What's your name?")
```




```
>>> print "Hello " + myName
Hello John
```

Task 6b: Error Trapping (specific)

Task6b.py – Catch the error before it's a problem

Check to see that the user date returns at least one record.

If not, indicate no records found…

Otherwise, proceed with the successful message

```
64    # Check that at least one record was returned
65  - if len(keyList) == 0:
66        print "No observations made on " + userDate
67  - else:
68        # Now that we have a list of keys, we can loop through them, extract the l
69        print "At " + userDate + ", Sara the turtle was found at:"
70  -     for k in keyList:                                    #Loop through the keys
71            userLoc = locationDict[k]                        #Get the lat/long tupl
72            print " Lat: "+userLoc[0]+"; Lon: "+userLoc[1]   #Print them to the scr
```

Anticipated errors can be dealt with somewhat explicitly

Error Trapping in Python

http://docs.python.org/release/2.6.5/tutorial/errors.html#handling-exceptions

8.3. Handling Exceptions

It is possible to write programs that handle selected exceptions. Look at the following example, which asks the user for input until a valid integer has been entered, but allows the user to interrupt the program (using `Control-C` or whatever the operating system supports); note that a user-generated interruption is signalled by raising the `KeyboardInterrupt` exception.

```
>>> while True:
...     try:
...         x = int(raw_input("Please enter a number: "))
...         break
...     except ValueError:
...         print "Oops!  That was no valid number.  Try again..."
...
```

The `try` statement works as follows.

- First, the *try clause* (the statement(s) between the `try` and `except` keywords) is executed.
- If no exception occurs, the *except clause* is skipped and execution of the `try` statement is finished.
- If an exception occurs during execution of the try clause, the rest of the clause is skipped. Then if its type matches the exception named after the `except` keyword, the except clause is executed, and then execution continues after the `try` statement.
- If an exception occurs which does not match the exception named in the except clause, it is passed on to outer `try` statements; if no handler is found, it is an *unhandled exception* and execution stops with a message as shown above.

A `try` statement may have more than one except clause, to specify handlers for different exceptions. At most one handler will be executed. Handlers only handle exceptions that occur in the corresponding try clause, not in other handlers of the same `try` statement. An except clause may name multiple exceptions as a parenthesized tuple, for example:

Task 6c: Error Trapping (general)

Task6c.py – Catch any error before it's a problem

Indent error prone text under a *try:* clause. If any error occurs within the section under the *try:* clause, Python will skip to the *except:* clause.

Raise an error if the date supplied is not in the right format. Error trapping is triggered.

Raise an error if no records are returned

If no errors occur, the *except:* clause is skipped.

```
53  try:
54      # Ask the user to enter a start record and and end record
55      userDate = raw_input("Enter date of record (M/D/YYYY):")
56
57      if "/" not in userDate:
58          raise Exception(userDate + " is not a valid date format")
59
60      # Create a list of all the dictionary keys (UIDs) with a date ma
61      keyList = []                    #Create an empty list to which we
62      for k in dateDict.keys():       #Loop through all the keys in the
63          v = dateDict[k]             #Get the value corresponding to th
64          dateValue = v[0]            #The current value is a date, time
65          if dateValue == userDate: #Check whether the date matches th
66              keyList.append(k)       #If it does, then add the key to t
67
68      # Check that at least one record was returned
69      if len(keyList) == 0:           #If
70          raise Exception("No observations made on " + userDate)
71      else:
72          # Now that we have a list of keys, we can loop through them,
73          print "At " + userDate + ", Sara the turtle was found at:"
74          for k in keyList:                                       #Loop th
75              userLoc = locationDict[k]                           #Get the
76              print " Lat: "+userLoc[0]+"; Lon: "+userLoc[1]   #Print t
77
78  except Exception as e:
79      print e
```

Unanticipated errors are handled generally

Writing scripts

- Approach a scripting project by mapping out the logical flow what you want to accomplish
- Construct your script in incremental steps
- Include comments throughout your script
- Give useful names to your variables

Writing scripts gets easier with experience and more knowledge of the scripting language

QUESTIONS?