Introduction to Python for Digital Mapping

Vincent A. DiNoto, Jr., GISP

Director of the GeoTech Center

Vince.dinoto@kctcs.edu

Introduction

Python is a scripting language that can be used with several different mapping programs such as ArcMap Desktop, ArcGIS Pro and QGIS; each program has a different mapping library. There are differences between 32 and 64 bit versions of Python. The script cannot easily be transferred from one software package to another, for this workshop ArcMap Desktop will be the mapping software for which scripts will be developed.

This workshop is based on a course that was developed by the GeoTech Center and is offered to students at Jefferson Community and Technical College. The Jefferson course, GIS 255, website is http://pythongist.weebly.com/ the model course information can be downloaded from http://geotechcenter.org/. Data for the workshop will be provided to the participants. The course at Jefferson is offered in three different formats, online, in person and learn on demand. It is assumed that participants will have reviewed the first few chapters of the online text.

The Pythonwin (IDE) editor will be used in this workshop, there are other Python editors that could be used. In general, all operations will be done in programming mode, immediate mode will not be used. Python 2.x and ArcMap 10.5.x are used. ArcMap must be installed on the computer for the Python mapping to function properly, thus this content will only work on Windows based operating systems. ArcMap needs to be installed prior to installing of the IDE.

Formatting is critically important, variables will be either text or a number. Text (string) can contain numbers, but mathematics cannot be performed, while a number variable allows for mathematical functions to be applied. String values always use double quotation marks around them (“ “).

Getting Started

Comments should be made throughout the code to assist in editing and troubleshooting the code, a pound sign (#) is used in front of the descriptions, the comments can be on their own line or after a function. Arcpy is a module contained within ArcMap, this is the mapping library of geoprocessing functions that will be used in Python. The library should be loaded as the first line after comments:

```
import arcpy
```

The code is case sensitive. The pathways to data and storage locations are critical and will first be shown as direct paths, an r is used with the pathway so standard formats can be used. For example:

featureOutput=r"C:\Users\vdinotojr0001\gis data\KY.gdb\college"

Note the file college is located in the KY geodatabase, if located in a folder the programmer would be required to put the shp extension on the pathway of the file. Note: the variable featureInput would be a different variable if spelled featureinput. The results file might look like:

featureOutput= r"C:\Users\vdinotojr0001\gis data\KY.gdb\buffer"

Thus, the output file is being saved in the same geodatabase as the input data. Note if the file name buffer would have already existed in the geodatabase, an error would be generated since the file being saved does not have overwriting permissions.

The Buffer command would take the following format:

arcpy.Buffer_analysis(Input, Output, Radius)

The arcpy library is being used, if it had not been imported at the beginning of the scrip, you could not use the buffer command. The Buffer command and the associated toolbox must be specified and separated by an underscore. In ArcMap the Buffer command is located in the analysis toolbox. The first parameter is the data file which will be buffered. The input feature could be a variable or a direct pathway to the file. The second parameter is the storage location of the created buffer, which could be a direct pathway or a variable. The final required parameter is the radius of the buffer, note it must specify the units as part of the radius and can be a variable or directly inputted. The data needs to be projected for the buffer to work properly just as in ArcMap. Quotation marks are used around the value of the variables because this information is a string.

Therefore, the total script would take a format similar to this:

```
import arcpy
featureInput=r"C:\Users\vdinotojr0001\gis data\KY.gdb\college"
featureOutput=r"C:\Users\vdinotojr0001\gis data\KY.gdb\buffer"
radius="25 miles"
arcpy.Buffer_analysis(featureInput, featureOutput, radius)
```

The script must be saved before it can be run in the IDE. If the script works properly a new file should be added to the KY geodatabase. Check to see if the file was created and then attempt to load the input and output file inside of ArcMap.

Environment (env)

Setting the environment allows for the programmer to create a shorter pathway in the variable definitions. In general, the environment statement is placed as one of the first lines in a program. Using the previous script, a modification will be made so it can use the environment statement and additional variables.

```
import arcpy
from arcpy import env
env.workspace = r”C:\Users\vdinotojr0001\gis data\KY.gdb”
featureInput=“college”
featureOutput=“buffer1”
radius=”25 miles”
arcpy.Buffer_analysis(featureInput, featureOutput, radius)
```

Note the script is much simpler. The name of the output feature was changed since buffer would already exist in the KY geodatabase.

Clip

A county boundary file is being used to clip rivers, roads and landmarks out of state files. The clip command takes the following format:

Arcpy.Clip_analysis(input feature, boundary file, output feature)

The clip feature is located in the analysis toolbox. A command can be used multiple times in the same script, so therefore to perform three clips the same command would be used three times. Variables should be used whenever possible. Example code:

```
import arcpy
from arcpy import env
env.workspace = r”C:\Users\vdinotojr0001\gis data\KY1.gdb”
county=”Trimble”
arcpy.Clip_analysis(“kyrivers”,”KY_Trimble”,county+”_rivers”)
```

The first three lines of code are the same as before except that the information is in a different geodatabase. The next line of code is defining a variable called county, note the double quotes since this is a string. The final line of code is calling the clip tool. The file to be clipped is the kyrivers file which could have been a variable for the input. The next term is the polygon file that is the clipping boundary. Finally the last expression is using the county variable with a string and summing them together to create a new file called Trimble_rivers, which will be saved in the KY1 geodatabase. The clip expression could be repeated for other state items such as roads. Note that no extension is required on any of the files since they are stored in a geodatabase. If the output file has been created previously and stored in the geodatabase an error will be generated.

Merge

The merge command creates a new file by combing like type of files together. The files to be combined together will be placed in a list variable (array). Only those files of a like type should be placed in the array but the files need not represent contiguous surfaces. Example code:

```
import arcpy
from arcpy import env
env.workspace = r”C:\Users\vdinotojr0001\gis data\KY1.gdb”
listCounty=[“KY_Jefferson”, “KY_Oldham”, “IN_Clark”]
arcpy.Merge_management(listCounty,”IN_KY_Merge”)
```

The first three lines of code are the same as used in the previous example. The fourth line of the code is defining a new list variable, it is a list because it uses square brackets. Each element of the variable has double quotes since they are a string function. The merge command is in the management toolbox. The form of the command is the list of items to be merge as the first variable (the input) and the second variable is the name of the resulting output file. Multiple merges can be performed in a single script and using good descriptive variables is also an important consideration.

Note that good naming convention is extremely important in all of geospatial technology but particularly important in Python scripting. For example, using the state and a short description of what is created, like KY_roads.

If

The if statement allows the designer to make different branches depending on the results of the logic statement. If is used in conjunction with the elif statement and else statement. The if statement asks a question and if true a single or multiple operations are performed, if false it branches to the elif (else if) statement and another logical question is asked of the data, when true others operations are done and when false it can branch to additional elif statements or an else statement. In Python it is important to indent the script to separate the decisions that are being made, without the same amount of indentions the code will not function properly. Most IDEs will automatically indent after the if statement. If statements can be nested within other if statements.

Labels

This component will be discussed from within ArcMap and the label command. The file that will be used in this exercise is the KY_Railroad file. Open ArcMap and load the KY_Railroad file, then open the properties window and select the labels tab. Once the labels tab has opened select the expression button. Check the advanced box, make sure the parser is Python and select the RROWNER field. Since the designer is working within ArcMap and has selected a specific file, the environment and arcpy statements need not be loaded. The software will start with a couple of lines of code, which will define the field to be explored, note this is a list variable. In this example, abbreviations in the RROWNER field will be changed to full names on the map. An if statement is used.

```
if rr==”CN”:
```

The format of the if statement is that a comparison is made with a single value from the RROWNER field to see if is CN and only if it is CN (the double equal sign). The colon is always placed at the end of the expression. When true, the items below the if statement which were indented, (this text will not auto indent so the designer must indent and always use the same number of spaces) are executed. The true expression is shown below:

```
return “Canadian National”
```

The return statement must be indented, instead of the label being CN on the map it will now be Canadian National, the original file has not been modified. If the expression was false then it would drop to the next line that is at the same indention level as the original if statement and generally will be an elif statement like the following:

```
elif rr==”PAL”:
```

The elif statement (at the same indention level as the if) does another comparison for when the list variable is PAL and only PAL it drops to the next indented line of script:

```
return “Paducah and Louisville”
```

Which would return to the map the full name of the PAL railroad. If the expression is false it would drop to next line that was not indented and execute the else statement (note if more railroads are to be named there would be another elif statement and not the else statement). The else statement is at the original indent level with the colon at the end. Indented under the statement is another return statement which is done if all the previous statements are false.

```
else:

return “Unkown”
```

Generally the final return statement is automatically loaded for the user and is at the original indent level.

```
return [RROWNER]
```

The total code looks like the following:

```
rr=[RROWNER]        #general auto inserted
if rr==”CN”:
  return “Canadian National”
elif rr==”PAL”:
  return “Paducah and Louisville”
else:
  return “Unkown”
return [RROWNER]
```

More advanced labeling is done by changing the size of the font, color, etc. to accomplish this CSS codes are used inside of diamond brackets (<>). For example the return might be modified like the following:

```
return “<CLR red=’255’><FNT size=’14’>”+”Canadian National” +”</FNT></CLR>”
```

The CLR command is for Color and red was the selected color with values between 0 and 255. If a second color is mixed with the first it would appear like green=’100’ with no commas between. The three color names are red, green and blue (RGB), which will give 65,000 unique colors. The CMYK system can also be used with ranges of 0 to 100. For more understanding of colors http://rapidtables.com/web/color/Web_Color.htm?T1=255+255+255#color%20table The FNT stands for font, the font face and size can be modified. Other commands such bold (BOL) can also be used. The first command started must be the last command closed. A command is closed by using the / and the command name such as closing of CLR with </CLR>. Therefore, first command opened is last closed and the next opened is the next closed and so on.

Field Calculator

Python scripts can be run in the field calculator, which can be opened through the attribute table. A new blank field must be defined before creating the script. In the field calculator select Python and check the Show Codeblock. There are two windows, the lower box will contain the actual field names, which are generated by using the list of fields, it is important that these names are selected and not typed. The upper textbox is where the code is created. The first line of the code is using common names that correspond to the actual field names in the lower box and need to have a one to one correspondence. So the definition function might look like:

```
def fnInformation (Type, Classes, Cert, Associate, Bachelor, Graduate)
```

The lower box must have six field values and have the same function name.

At this point if statements are used to select information. In this example, a nested set of if statements will be used. The first if statement takes the following form:

```
if Type==”2 Year College”:
```

So when the if state is true it will perform the commands placed below it, when false it will drop to the next statement line that has the same indention as the if statement, which in this case will be an else statement, that has the following format:

```
else:

return “4 year institution”
```

The return statement must be indented, so if it is not a 2 year college it must be a 4 year college or university. If the types are not one of the two listed types then other inaccurate results may occur.

When the if statement is true another set of if, elif and else statements are used. Like the following:

```
if (Classes==”Yes” and Cert==”Yes” and Associate==”Yes”):
```

A logical and is used which requires that all elements are true to give a true result. When true the code will go to a return statement which will state that it is a 2 year college offering classes, certificates and degrees, which will be added to the newly created column. When false it will drop to an elif statement.

The total code for the problem would look like the following:

```
def fnInformation (Type, Classes, Cert, Associate, Bachelor, Graduate)
if Type==”2 Year College”:
        if (Classes==”Yes” and Cert==”Yes” and Associate==”Yes”):
                return “2 year college offering classes, certificates and degrees”
        elif (Classes==”Yes” and Cert==”Yes”):
                return “2 year college offering classes and certificates”
        elif (Classes==”Yes” and Associate==”Yes”):
                return “2 year college offering classes and associate degree”
        elif Classes==”Yes”:
                return “2 year college offering only classes”
        else:
                return “2 year college offering no GIST”
else:
        return “4 year institution”
```

For

In the discussion of the for statement we will also discuss a new command called ListFields.

A for statement can take several different forms but one of the most common forms is to use a list of information and do the same command multiple times for each item in the list. The format takes the following form:

```
for item in list:
```

The first time the for statement is executed, the first item from the list variable will be the value of the item variable. The next line of the script will be indented because of the colon at the end of the statement. Once the end of the indent lines is reached, the script will return to the for statement and see if there are unused members of the list variable, if so, this will become the new value of the item variable and the indented functions will be repeated, this is repeated until each member of the list variable has been used.

The code takes the following format:

```
Import arcpy

From arcpy import env

Env.workspace=”P:\GIS_data\gis data\KY2.gdb”

fieldName=arcpy.ListFields(“KY_Rail”)

for field in fieldName:

        print field.name
```

The first three rows are common with what has been done on previous exercises. The fourth line uses the ListFields function which creating a list variable of all the fields in the KY_Rail file and storing in the list variable fieldName. The next line uses the for statement which is looking at each member of the list variable fieldName, the first time the variable field will contain the first member of the list and then the name will be printed on the screen. The script will then loop back to the for statement and a new value will be placed in the variable field. This will continue until all members of the fieldName list has been used.

There are many more arcpy commands and much more complicated functions in tools that can be used. Many additional functions are included in the class website.