JANUARY 13, 2017

MORE PYTHON

TOPIC 8

VINCENT A. DINOTO, JR.
JEFFERSON COMMUNITY AND TECHNICAL COLLEGE
Vince.dinoto@kctcs.edu

Contents

Table of Figures

Introduction

The purpose of this module will be to explore additional commands and functions of Python in ArcMap for Desktop. Some of the commands and method will be more complicated than those discussed to this point and will assume the knowledge of previous modules. In this module, the IDE will generally be used as the script editor working in the program mode. Other methods could be used, but only one will be demonstrated. All of the operations shown and discussed will require the use of the arcpy module. This module will cover more functions than covered in the previous modules since the foundations have now been set.

Functions to be Covered

- The **cursor** function has several different types, which include search, insert and update.
- Creating shapefiles without having ArcMap or ArcCatalog opened will be covered.
- Fixing broken links, which is a problem that is caused when files are moved.
- The mapping module will be covered which deals with final map creation.
- Exporting images and Adobe Acrobat files (pdf) as well as multiple exports for creating a map book.
- The layout of a map outside of ArcMap Desktop will also be cover.
- In addition, the **with**, **del** and **da** commands will be explored.

Del

The **del** statement will delete a variable and its content. When a variable is created, it is available even after the program has been completed. For example, if a variable containing your name was part of an executed script and the user changed to immediate mode and prints the variable it would still contain your name. It is always a good practice to delete the variable when they are no longer needed. It can also insure that information is not passed between different code steps, which would create inaccurate information. In addition, the **del** statement will release the locks on the file. The file is not being deleted from the hard drive, just from the program memory. If the **del** command is not used the variable may remain locked, i.e. such as a shapefile and not be able to be used in any other operations. The **del** command will be demonstrated in other pieces of code in this module and not as an independent command.

Da Module (Data Access)

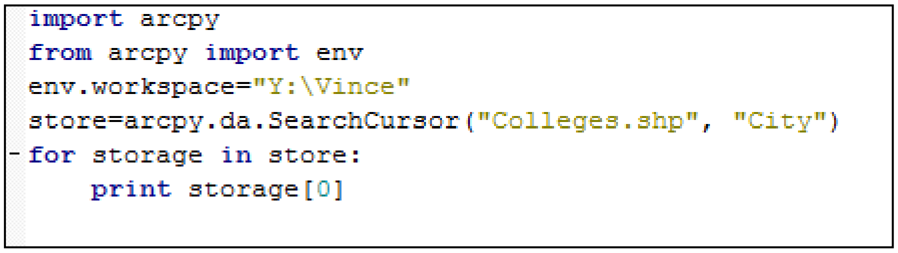
The **da** module is the location of the cursor command that will be explored in this module. It works similarly to the arcpy module, and must be imported prior to using the functions contained within the module.

Cursor

The cursor functions allow the user to work with a specific row in the attribute table. Each row is called a record. In many ways, the cursor function is similar to the **list** function discussed previously, which was used to control files and fields (columns, attributes). There are three cursor functions that will be explored, the **insert**, **update** and **search**. For the **cursor** function to be used an input table from a shapefile or a feature class file (in a geodatabase) containing field names.

Search

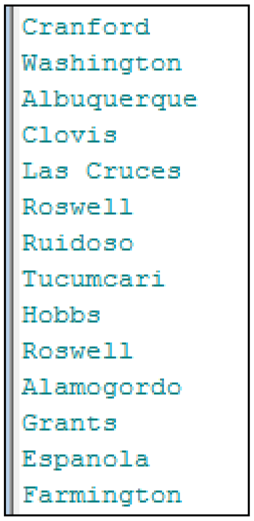
The **search cursor** allows the user to explore the contents of a shapefile or feature class file without opening it in ArcMap Desktop.



```
import arcpy
from arcpy import env
env.workspace="Y:\Vince"
store=arcpy.da.SearchCursor("Colleges.shp", "City")
for storage in store:
    print storage[0]
```

Figure 1: Search Cursor Script

1. In Figure 1, the arcpy module is imported and then the environment workspace is set to be the author's Y drive in a folder named Vince (remember a different path will be needed on the learners workstation).
2. A variable '***store***' is created for use with the **SearchCursor** command.
   a. The first parameter of the command is the shapefile (feature class for the geodatabase) that will be searched.
   b. The second parameter is the field that will be examined.
   c. To use this function, the user would need to know the field structure of the shapefile, which could be explored as noted in previous modules. Each value in the field City would be saved in the '***store***' list variable.
3. To output the results of the '***store***' variable a **for** loop has been set-up which will print each member of the City column one line at a time. In Figure 2, only a part of the list shown. All values are printed which includes duplications in different rows.



```
Cranford
Washington
Albuquerque
Clovis
Las Cruces
Roswell
Ruidoso
Tucumcari
Hobbs
Roswell
Alamogordo
Grants
Espanola
Farmington
```

Figure 2: Results of Search Cursor

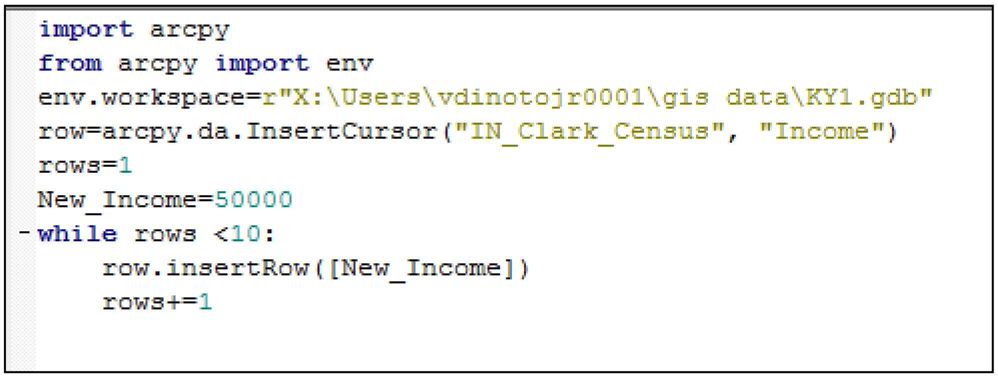
Assignment 8.1

Use the college shapefile that was presented in the demonstration in this lesson. Instead of just returning a single Column of information return the **city**, **state** and the name of the **college**. Additional points are awarded for good formatting of the information. Since there are more than 3000 records, only a portion of the results need to be sent to your instructor. Also include the code used to create the results, (there are multiple ways to accomplish this task) and additional points may be award for creativity as well.

Insert

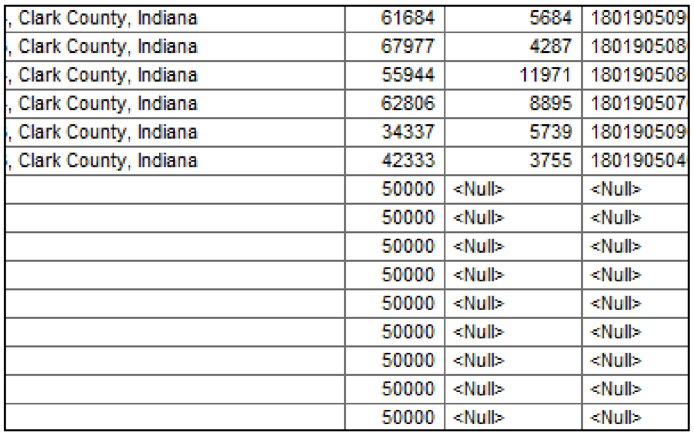
The **insert cursor** command allows the programmer the ability to insert a row into a shapefile or feature class file, the user does not have to have the file open. To input a single row a loop function would not be used in this example a new statement is being inputted multiple times using the **while** function, which will be fully explained in the code discussion.

1. Arcpy is imported and the environment is defined, a geodatabase is used for this example.
2. A variable '***row***' is defined which uses the **InsertCursor** command, which is part of the data access module (thus the reason that the data is added into the function statement).
   a. The first parameter is the file in which the new rows will be inserted
   b. The second parameter is the column (field) to write the value
3. The next two lines of the script define a counter variable called '***rows***' and the input value to be written to the Income column.
4. The **while** command is used note the logic and the colon at the end of the line. The loop is completed as long as the variable ***rows*** are less than 10.
5. The value of the variable ***New_Income*** is inserted into the Income column each time the loop is executed.
6. At the end of the loop the variable ***rows*** are incremented by one and the loop is repeated until the logic is no longer true.



```
import arcpy
from arcpy import env
env.workspace=r"X:\Users\vdinotojr0001\gis data\KY1.gdb"
row=arcpy.da.InsertCursor("IN_Clark_Census", "Income")
rows=1
New_Income=50000
while rows <10:
    row.insertRow([New_Income])
    rows+=1
```

Figure 3: Inset Cursor Script



, Clark County, Indiana	61684	5684	180190509
, Clark County, Indiana	67977	4287	180190508
, Clark County, Indiana	55944	11971	180190508
, Clark County, Indiana	62806	8895	180190507
, Clark County, Indiana	34337	5739	180190509
, Clark County, Indiana	42333	3755	180190504
	50000	<Null>	<Null>
	50000	<Null>	<Null>
	50000	<Null>	<Null>
	50000	<Null>	<Null>
	50000	<Null>	<Null>
	50000	<Null>	<Null>
	50000	<Null>	<Null>
	50000	<Null>	<Null>
	50000	<Null>	<Null>

Figure 4: Results

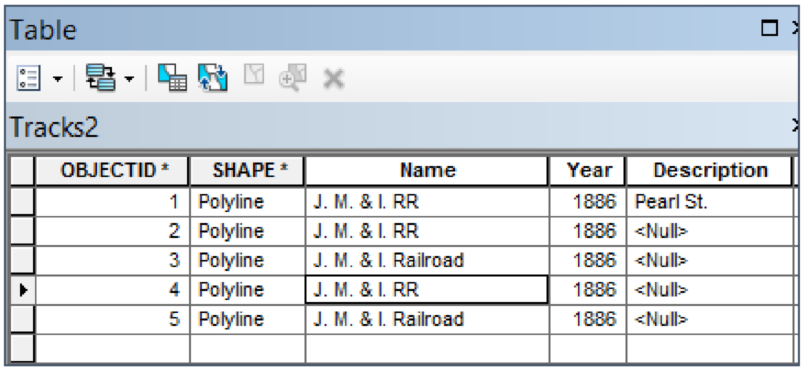
See Figure for the results of the process, note that only one field had data inserted which is not make a functional data structure, but does demonstrate the process.

Assignment 8.2

In this assignment, the student is to add 10 rows of information in three different fields. Send your instructor the output information and the script. There are multiple different ways to accomplish this task and any data file can be used to append the information. Additional points will be awarded for making the process as simple and useful as possible.

Update

The **Update Cursor** command will allow the user to change a value in a field or if the field is blank place a value into the field, thus no new rows are added like in the previous example. This function will also allow the user the ability to delete a row or rows of information. In general, a looping function will be used with this operation, so that the values can be updated in each row requiring a change. This command along with other commands could create a new field. For example, this method could automate the process performed in a tabular join that creates a new field with the appropriate format, so that the joining process with the spreadsheet can be accomplished. It is extremely important that the shapefiles or the feature class file is not open by any user. That would create file locks and most likely would prohibit the operation. The table in Figure 3 is part of an attribute table from a historical map that was digitized from a Sanborn map of the 1880s of Jeffersonville, Indiana. Examining this data shows that the individual who performed the digitizing did not use a domain when naming the railroad, since the same railroad has two different names. If a user attempted to query for the J. M. & I Railroad (Jeffersonville, Madison and Indianapolis), they would only get part of the record because of the two different formats the railroad is listed. The Update Cursor will be used to make all the names the same.

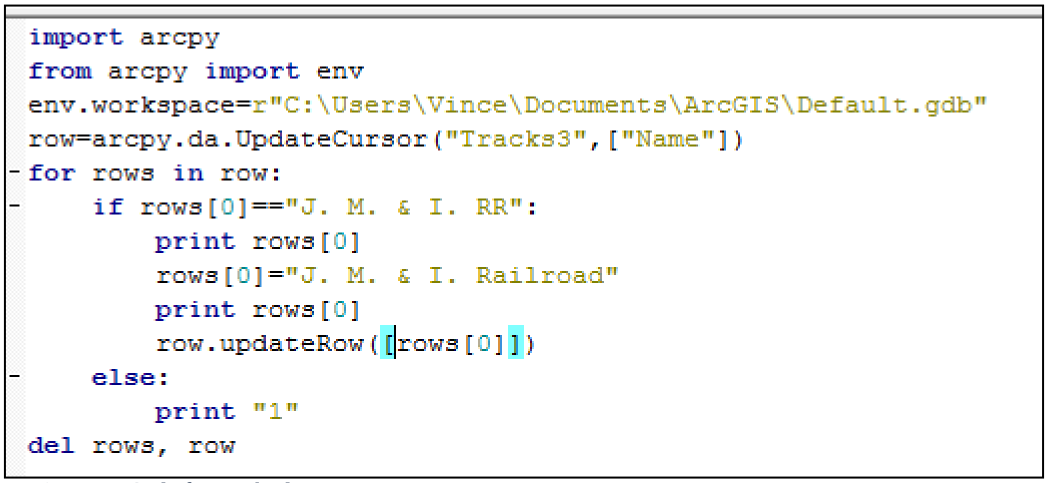


OBJECTID *	SHAPE *	Name	Year	Description
1	Polyline	J. M. & I. RR	1886	Pearl St.
2	Polyline	J. M. & I. RR	1886	<Null>
3	Polyline	J. M. & I. Railroad	1886	<Null>
4	Polyline	J. M. & I. RR	1886	<Null>
5	Polyline	J. M. & I. Railroad	1886	<Null>

Figure 3: Attribute Table

Three different methods will be shown using the same data. Remember if a copy of the data is not made, after the first method runs successfully it will change the data. Thus the errors will no longer exist and cannot be successfully used for the next two demonstrated methods.

Method 1

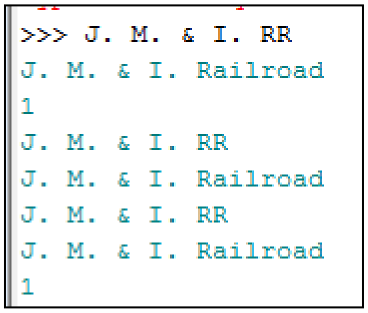


```
import arcpy
from arcpy import env
env.workspace=r"C:\Users\Vince\Documents\ArcGIS\Default.gdb"
row=arcpy.da.UpdateCursor("Tracks3",["Name"])
for rows in row:
    if rows[0]=="J. M. & I. RR":
        print rows[0]
        rows[0]="J. M. & I. Railroad"
        print rows[0]
        row.updateRow([rows[0]])
    else:
        print "1"
del rows, row
```

Figure 6: Code for Method 1

Figure 6, shows the method 1 scrip and is discussed below.

1. Arcpy is loaded and the environment is set to the geodatabase.
2. The variable ‘***row’*** is defined with the **UpdateCursor**. The first parameter is the file name and the second parameter is the field. Note the field name is in square brackets, indicating that a list has been defined.
3. The **for** loop will be used for the operation with an **if** statement within. The **for** loop will ensure that each member of the list variable ***row*** is examined. The variable ‘***row’*** contains all the members from the field **Name** and each record is examined in the **Name** field, which will be the same item modified if required.
4. The **if** statement is comparing the value of the individual cell and determining if it is true or false by what is contained in the quotation marks.
5. If it is false it drops to the **else** statement, prints a number 1 and returns to the **for** statement.
6. If it is true, it prints the current value in the cell,
   a. Redefines the cell to have a new value,
   b. Prints the new value
   c. Stores the value into the cell.
7. The **del** command was used to delete the variables ***rows and row***.

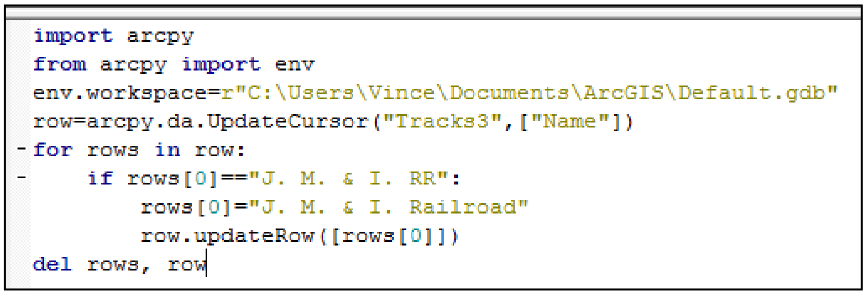


```
>>> J. M. & I. RR
J. M. & I. Railroad
1
J. M. & I. RR
J. M. & I. Railroad
J. M. & I. RR
J. M. & I. Railroad
1
```

Figure 4: Output from Method 1

Figure 4 shows the output in which three items were modified and two were unchanged.

Method 2



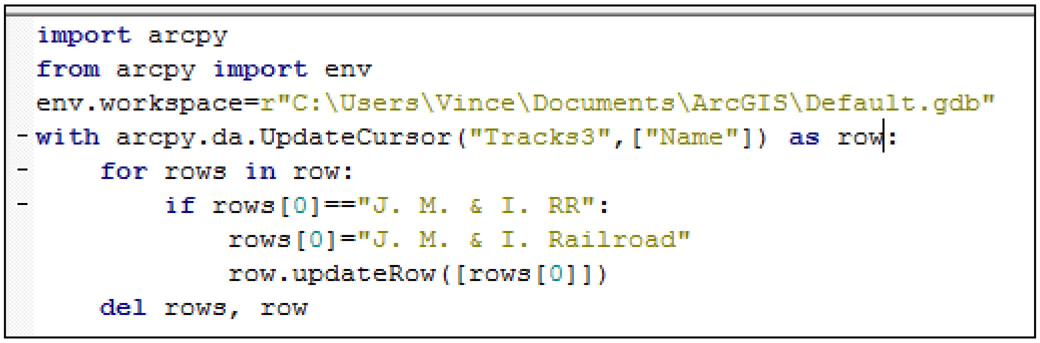
```
import arcpy
from arcpy import env
env.workspace=r"C:\Users\Vince\Documents\ArcGIS\Default.gdb"
row=arcpy.da.UpdateCursor("Tracks3",["Name"])
for rows in row:
    if rows[0]=="J. M. & I. RR":
        rows[0]="J. M. & I. Railroad"
        row.updateRow([rows[0]])
del rows, row
```

Figure 5: Method 2 code

In the second method shown in Figure 5, the print statements have been removed which shows how the code flows, this code would be more appropriate for production level work, while the previous code is used for learning and troubleshooting. The data file was changed as a result of Method 1 and a new data file is required that contains the errors. To determine if this method if was successful the feature class file in the geodatabase would need to be opened in ArcMap (there are ways to determine this without opening the file).

Method 3 – with statement

In the third method the **with** statement will be used. The **with** statement is used to insure that all locks have been removed from a dataset once the operation has been completed. As noted earlier when certain operations are performed datasets are locked so that no other user can modify the data while being used. When the operation is executed, it is critically important to remove the locks so that the datasets are available to other users. This is of particular importance for files in a geodatabase, but these discussions do not refer to files contained in a relational database. The code shown in Figure 6, uses the **with** statement. The **update cursor** statement is part of the **with** statement and ends with a colon. This will cause all lines after this statement to be indented. The file will be locked until the script returns to the initial indentation level, which will then unlock the file.



```
import arcpy
from arcpy import env
env.workspace=r"C:\Users\Vince\Documents\ArcGIS\Default.gdb"
with arcpy.da.UpdateCursor("Tracks3",["Name"]) as row:
    for rows in row:
        if rows[0]=="J. M. & I. RR":
            rows[0]="J. M. & I. Railroad"
            row.updateRow([rows[0]])
    del rows, row
```

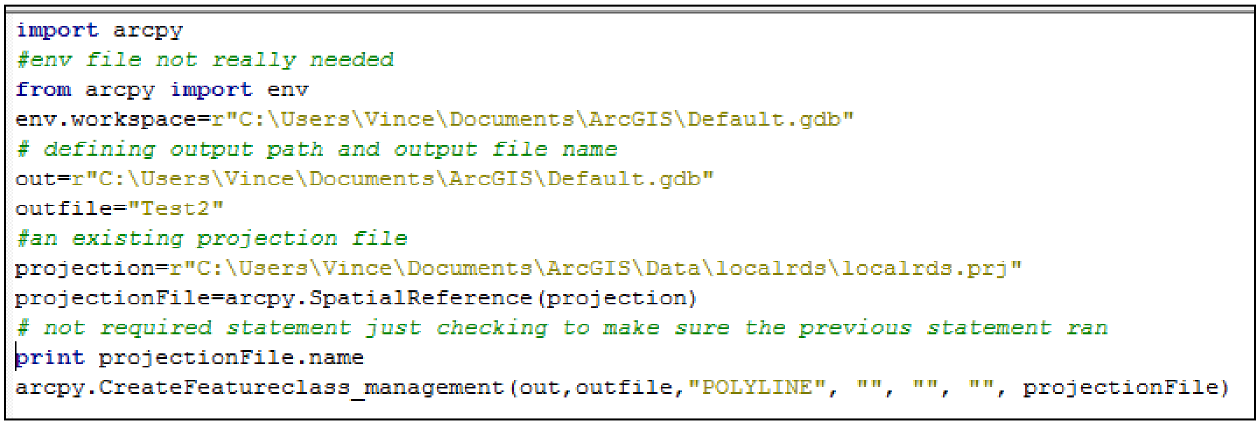
Figure 6: Using with Statement

This example is basically the same code as in Figure 5. It has been slightly modified to use the **with** statement. The only changed line is the fourth line from the top. Defining the variable row is done differently using the **with** statement and the colon at the end. The '***as row***' variable is used instead of the equal sign as before, the equal sign would cause an error to occur. Note all the lines are indented after the **with** statement and since the script ends after the **del** statement, there is no returning back to the original indention. This method will insure that the file closes at the end of the operation.

Assignment 8.3

A file has been created that did not use a Domain; this allowed users input Yes and No answers in different formats. Therefore, the data table is not uniform and a query cannot be successfully performed. The file contains yes, YES, Y, no, NO, and n. The correct format should be Yes and No. The task is to fix the file so that all Yes and No answers are in a single format. Provide your instructor with the script and screen shot of the file before and after the script has been run.

Shapefile Creation



```
import arcpy
#env file not really needed
from arcpy import env
env.workspace=r"C:\Users\Vince\Documents\ArcGIS\Default.gdb"
# defining output path and output file name
out=r"C:\Users\Vince\Documents\ArcGIS\Default.gdb"
outfile="Test2"
#an existing projection file
projection=r"C:\Users\Vince\Documents\ArcGIS\Data\localrds\localrds.prj"
projectionFile=arcpy.SpatialReference(projection)
# not required statement just checking to make sure the previous statement ran
print projectionFile.name
arcpy.CreateFeatureclass_management(out,outfile,"POLYLINE", "", "", "", projectionFile)
```

Figure 7: Creating a Shapefile

In this example, a shapefile will be created from outside of ArcMap and will have a projection. There will be no attributes created for this file. The construction of the shapefile/feature class file is done using an IDE in program mode. The **Cursor** and **List** commands were not used but could have been used to populate the shapefile. Therefore, what is created is an empty shapefile/feature class file. The tool used in this example is the **CreateFeatureclass** tool, which is located in the management toolbox and the **SpatialReference** command. An existing projection file (**localrds**) is used to set the projection for the new feature class file. This example will use a geodatabase as the storage location. The code is shown in Figure 7.

1. Arcpy is imported and the environment is set.
2. Three variables are defined; the third is the pathway to the projection file that will be used.
3. The fourth variable **projectionFile** defines the projection of the created file to be the same as that of the **localrds** file; it uses the **SpatialReference** command.
4. The **print** command is not required it is used only to show that the operation is working properly.
5. The final command is the **CreateFeatureclass**, which is a tool in the management toolbox.
    a. The first parameter is the pathway to where the new file will be created
    b. The next parameter is the name of the file that will be created

c. The third parameter is the type of file that will be created and for this case it is a line file
d. The next three parameters are not defined and thus are skipped; the final parameter is the projection type, which is defined in the ***projectionFile*** variable.

Case Study 1

In this case study, the learner will create a new shapefile that contains at least two fields and five rows. The fields should be populated with data. The data should have a map projection. The entire case study should be done without opening ArcMap or ArcCatalog. Provide the instructor with the script and the created shapefile or geodatabase containing the created file. Additional points award if the data can be displayed in ArcMap.

Fixing broken Links

One of the common problems that is experienced in geospatial technology is broken links to data files, which generally occurs when either the project file is moved to a new location, a different computer is used, or that the data files have been moved to a different storage location. In general, it is a slow process fixing one or more broken links, even when all the broken links are in the same folder or geodatabase. Remember that a broken link appears with a red exclamation point. If the file is selected there will be a check in the box, but no information will be displayed. Generally, this problem is fixed by individual re-linking each file. Figure 8, shows four broken links in the table of contents.

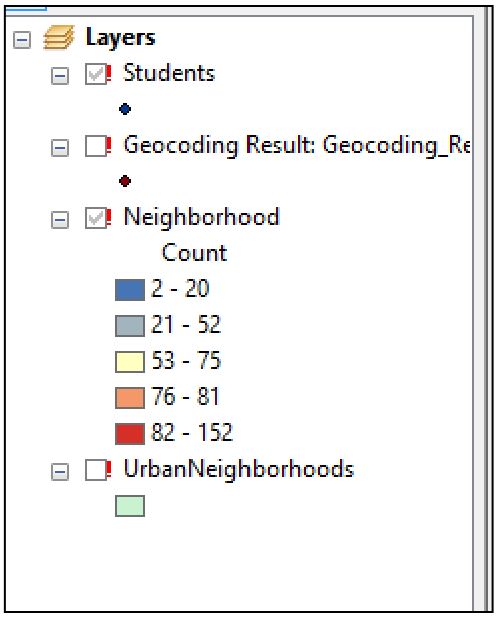


Figure 8: Broken Links

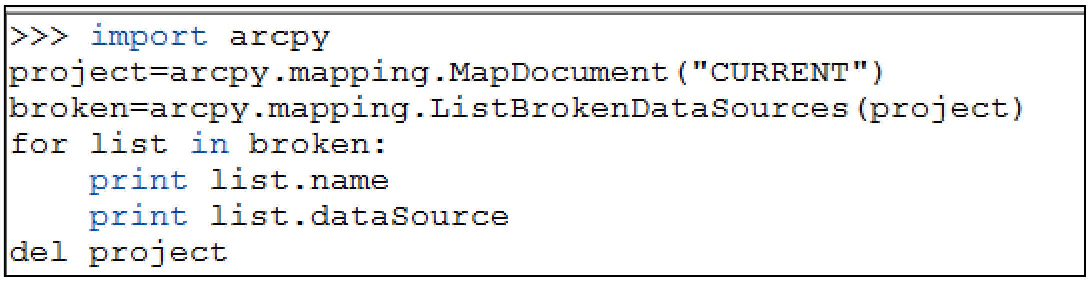
Mapping Module

The mapping module is used as part of the solution to fix broken links. For this example, the Python Window in ArcMap will be used and the project file (mxd) will be opened in ArcMap. The operation could be done using an IDE, but generally, the user realizes the problem once they open ArcMap. The word '**Current'** will be used in the script and this will only work when in ArcMap and the project file is opened. If the project file is not opened, then a qualified pathway to the information would be required.

Broken Links

First, a pathway to each shapefile that has a broken link must be determined; we know the name of the file with the broken link because that can be observed from the table of content, see Figure .

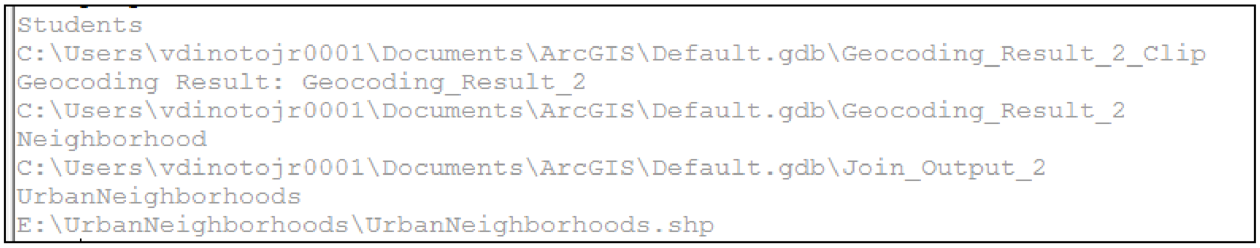
1. To determine the name of the file the **MapDocument** and **ListBrokenDataSources** commands are used. The **MapDocument** is using the parameter Current which not a pathway since the project file is open in ArcMap. The **MapDocument** command returns all the file names to the variable ‘***project***,’ i.e. the contents of the project file, which is a list variable.



```
>>> import arcpy
project=arcpy.mapping.MapDocument("CURRENT")
broken=arcpy.mapping.ListBrokenDataSources(project)
for list in broken:
    print list.name
    print list.dataSource
del project
```

Figure 12: Script for Broken Links

2. The ‘***broken’*** variable will contain only those files that have a broken link through the use of the **ListBrokenDataSources** command.
3. The **print** commands are used to display the name of the broken files and the pathway to the broken file links, see Figure 13.

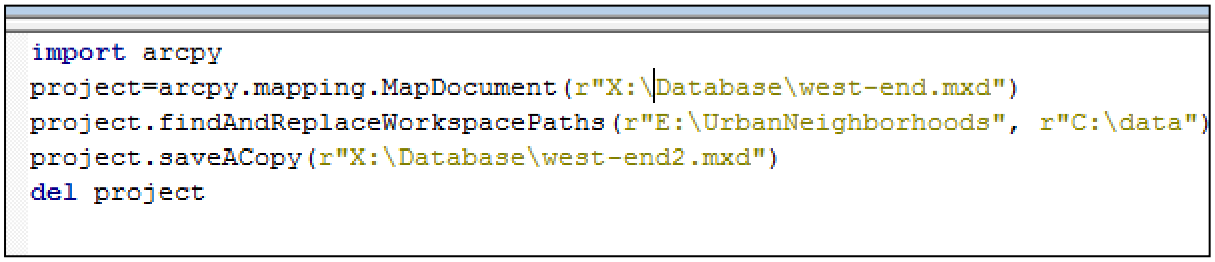


```
Students
C:\Users\vdinotojr0001\Documents\ArcGIS\Default.gdb\Geocoding_Result_2_Clip
Geocoding Result: Geocoding_Result_2
C:\Users\vdinotojr0001\Documents\ArcGIS\Default.gdb\Geocoding_Result_2
Neighborhood
C:\Users\vdinotojr0001\Documents\ArcGIS\Default.gdb\Join_Output_2
UrbanNeighborhoods
E:\UrbanNeighborhoods\UrbanNeighborhoods.shp
```

Figure 13: Results of the Script from Figure 12

Fixing Links

In the previous section, the broken links were determined but no repair was done. If the only purpose is showing where the project file data is located, can be useful but does not solve the problem. There are multiple ways that this problem can be solved and this example will show only one method. It will be different from the method shown Figure . The method shown for fixing the link will be performed in an IDE and thus the full pathways must be defined instead of using the word **current**. It is suggested that the project file (.mxd) be closed before beginning this process. See Figure 9, to follow the process.



```
import arcpy
project=arcpy.mapping.MapDocument(r"X:\Database\west-end.mxd")
project.findAndReplaceWorkspacePaths(r"E:\UrbanNeighborhoods", r"C:\data")
project.saveACopy(r"X:\Database\west-end2.mxd")
del project
```

Figure 9: Fixing Broken Link

1. Since this code is being created in an IDE the arcpy module must be added.
2. The variable ‘***project***’ defines the location of the project file.
3. The next line of script replaces the broken pathway with the correct one using the **findAndReplaceWorkspacePaths**. The first parameter is the pathway of the broken file and the second line is the pathway to where the file is currently located.
4. The **saveACopy** command then saves the mxd, while a new mxd is not required, the author believes by creating a new project, this reduces the chances of locks causing potential problems. Therefore, the mxd is saved with a number 2 on the end.
5. The final command deletes the variable project to get rid of any potential locks.

The results shown in Figure 10 from File Explorer and Figure 11 shows the table of contents of the new MXD, the west-end2.mxd file. The broken link is no longer shown for the UrbanNeighborhoods file. There is still a broken link for Neighborhood; because it had a different storage location than the UrbanNeighborhoods file.

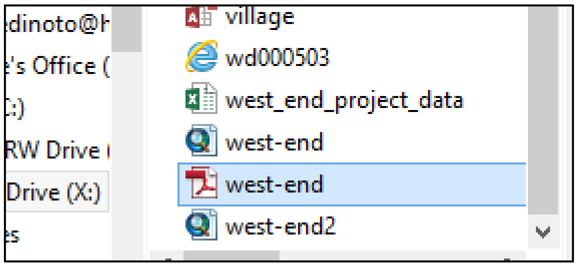
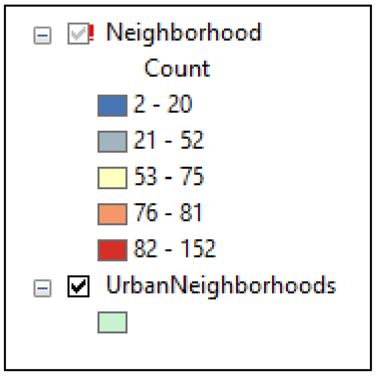


Figure 10: Results 1



Neighborhood
Count
🟦 2 - 20
▨ 21 - 52
🟨 53 - 75
🟧 76 - 81
🟥 82 - 152
UrbanNeighborhoods
🟩

Figure 11: Results 2

Assignment 8.4

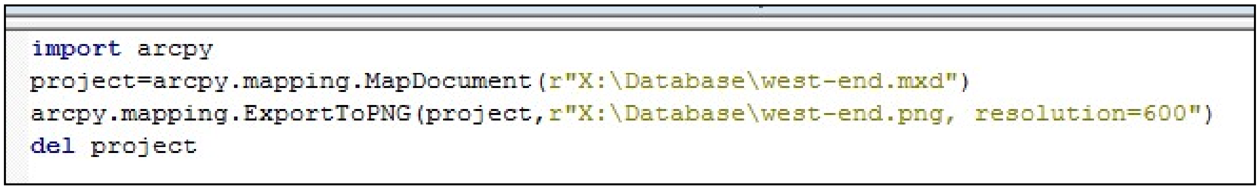
Write the required code which will fix two broken links or more, the broken links must be in two different storage locations. Change the links in the project file to the location where the data files are now located. Write this information to a new project file with a different name. Provide your instructor with the script and a map package containing both the old and new project files.

Outputting Maps

Once a map is constructed, the ability to output a publishable copy of the map is usually required, either to use as an image in a document or as a standalone item for plotting or printing. In my work, I usually need an image file that can be used in a report, a PowerPoint presentation or a poster presentation. Generally, an Adobe Acrobat (pdf) file will be created as a standalone map. Sending the final map directly to a server is not part of this discussion. In this section, the user will explore different electronic formats and resolutions for the output map. Create a map with proper formatting in Esri ArcMap, with items such as title, legend, text box, scale and directional arrow.

Exporting an Image File

In this example, the user will export a PNG format image file at a resolution of 600 dpi. The mapping module and a new command **ExportToPNG** are used to accomplish the task. The export operation only requires a few lines of code. This operation could be done inside or outside of Esri ArcMap in this case it was done in an IDE.



```
import arcpy
project=arcpy.mapping.MapDocument(r"X:\Database\west-end.mxd")
arcpy.mapping.ExportToPNG(project,r"X:\Database\west-end.png", resolution=600")
del project
```

Figure 127: Export to PNG

1. Arcpy is loaded
2. A pathway variable ‘***project***’ is defined. The project file contains the location of the mxd file.
3. The third line of code is the operational line that creates the PNG file. The **ExportToPNG** parameters are required for the input project file, the output name of the png and the resolution of the output in dots per inch (DPI). The output image file is defined within the command but could have been defined as a variable.

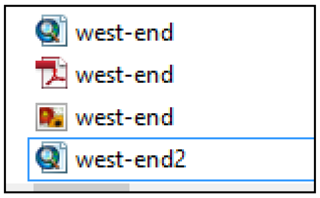
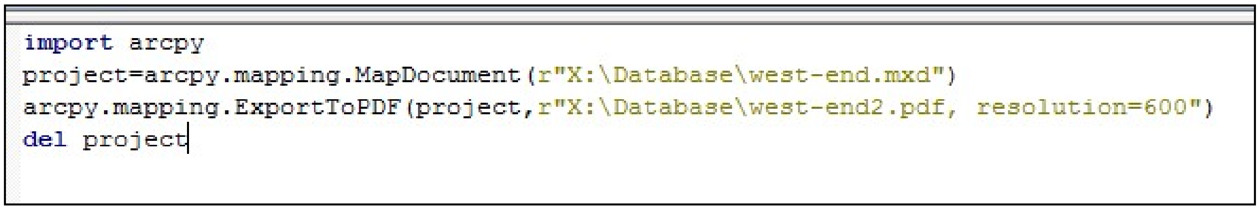


Figure 18: Results of Exporting a PNG file

4. The delete command is used with the ***project*** variable so that all locks are removed. The result of the code is the creation of an image file in the same location and with the same name as the original project file. See Figure for the results of the operation. Other image formats can be exported besides a PNG.

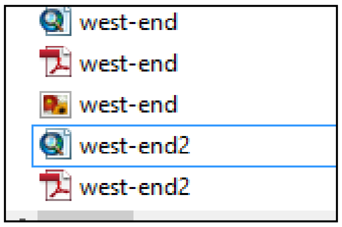
Exporting Adobe Acrobat File Part 1



```
import arcpy
project=arcpy.mapping.MapDocument(r"X:\Database\west-end.mxd")
arcpy.mapping.ExportToPDF(project,r"X:\Database\west-end2.pdf, resolution=600")
del project
```

Figure 19: Export PDF

Exporting a pdf file is similar to the previous discussion of exporting an image, see Figure . The only difference is in the export statement a different export command is used. All the notes for the lines of code in the image export are the same, except the command is **ExportToPDF**. In Figure note, a pdf file with the name west-end.pdf already exists from a previous project. The output name for the pdf file will be west-end2.pdf, see Figure and Figure . The user should not attempt to overwrite a file.



west-end
west-end
west-end
west-end2
west-end2

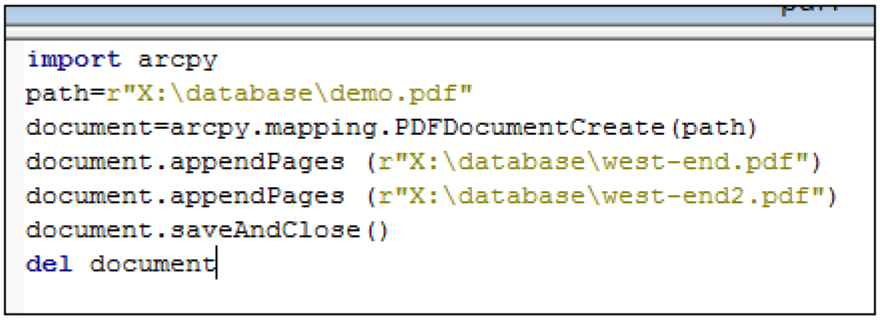
Figure 20: Results of PDF Creation

Assignment 8.5

In the assignment, the user will create a script that will do multiple exports of image and pdf files. The more efficient the code the higher the scored results will be once the operation is found to be correct. Provide the instructor with a screen shot of the storage location of the data before and after the script is run. The script should create files of the following types: PNG @600 dpi, JPEG @300 dpi, TIFF @450 dpi and PDF @600 dpi. The user should first create a final map before running the Python script.

Exporting an Adobe Acrobat File Part 2: Map Book Creation

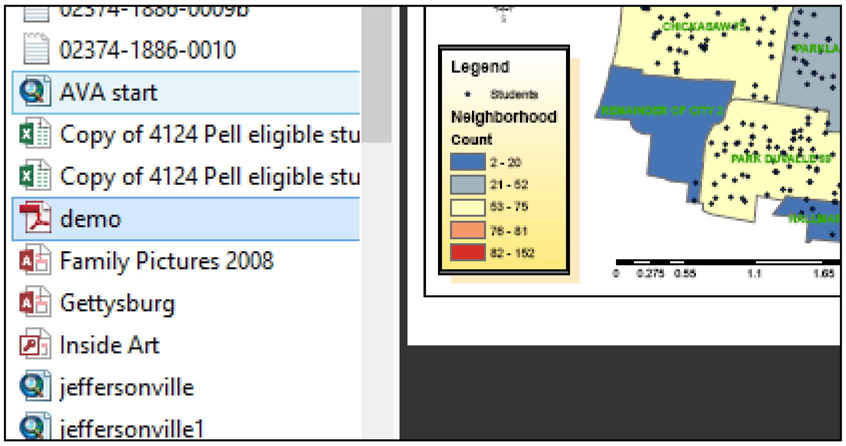
Many times the user will be required to create several maps of the same area. For a typical demographic research project, the user might need to make maps of Population, Income, Education Attainment, Race, Gender, Ethnicity, Housing Type, Geography (rivers and roads), Political Division (townships, towns, cities, place names). It is a time consuming process to manually create each file and then add them into a single pdf document. The Map Book command will create a single pdf containing multiple pdf files that were previously created. The pdf files could have been previously created as part of the same operation as discussed in Part 1.



```
import arcpy
path=r"X:\database\demo.pdf"
document=arcpy.mapping.PDFDocumentCreate(path)
document.appendPages (r"X:\database\west-end.pdf")
document.appendPages (r"X:\database\west-end2.pdf")
document.saveAndClose()
del document
```

Figure 21: Appending PDF FIles

1. The pathway to the first pdf file is established with the ***path*** variables other pdf files will be hard coded as parameters.
2. **ThePDFDocumentCreate** command is used and the parameter is the location to put the pdf file.
3. Two files are appended using the **appendPages** command to the original document created.
4. The last part of the appending is closing and saving the document with the **saveAndClose** command. No parameters are given to this command.
5. The final line of the script is the deleting of the ***document*** variable to insure that no locks will cause problems.



02374-1886-0010
AVA start
Copy of 4124 Pell eligible stu
Copy of 4124 Pell eligible stu
demo
Family Pictures 2008
Gettysburg
Inside Art
jeffersonville
jeffersonville1

Figure 22: Result of Map Book Creation

Remember, in this example the pdf files have already been completed. With a simple naming convention, the process could have been completed using a looping function. The pdf files could have been created as part of the script. The two pdf files are saved in the file, which was created in the third line of the script.

Assignment 8.6

In this assignment, a single regional geography will be used. The geography should be of a single county with census tracts visible. The map book created must contain at least five demographic maps. The pdf files and map book should be created as either a single script or two scripts. Provide your instructor with the script(s) and the resulting pdf file. Extra points will be awarded if a single script is used.

Case Study 2

Your boss has assigned you a project in which you must create final map documents that contain the follow.

1. A single county at the census tract level with tracts, major roads, and rivers displayed.
2. A population map of the county at the census tract level.
3. An income map of the county at the census tract level.
4. An African American population map of the county at the census tract level.
5. A white population map of the county at the census tract level.

You are informed that each map must contain a legend, title, directional arrow, scale bar and title block about who created the map. You must provide the boss the following products:

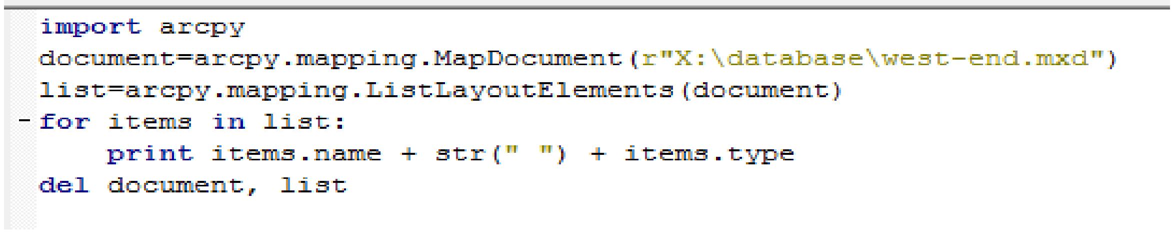
1. A pdf and a PNG file at 600 dpi resolution for each map.
2. As well as a map book that contains a cover page, (this was created in a word processor).

The maps can be created using ArcMap Desktop software and can be saved in five different project files (mxd). The items listed, as your products (1 & 2) must be done in a single script.

Map Layout

Without opening a project file it would be nice to understand the content of the project (mxd). This could save lots of time, especially when there are several project files that are being used and not sure which one contains the content needed. The user also doesn't have to wait for the software to install.

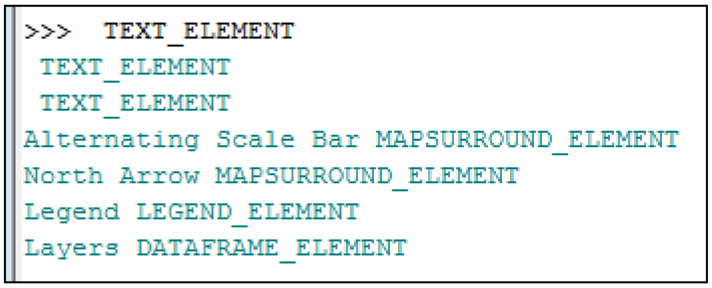
1. The variable ***document*** is defined as the west-end project file (mxd).
2. A variable named ***list*** is defined which will create a list variable of the type of items in the project file.
3. The command **ListLayoutElements** is used to load information into the variable.



```
import arcpy
document=arcpy.mapping.MapDocument(r"X:\database\west-end.mxd")
list=arcpy.mapping.ListLayoutElements(document)
for items in list:
    print items.name + str(" ") + items.type
del document, list
```

Figure 13: List of Layout Elements

4. A loop function is used to display the mapping elements, two parameters of the mapping elements are selected, the name of the element and the type of item. A space is placed between the two elements.
5. Once the loop has displayed the information, the variables are deleted to eliminate any locks.

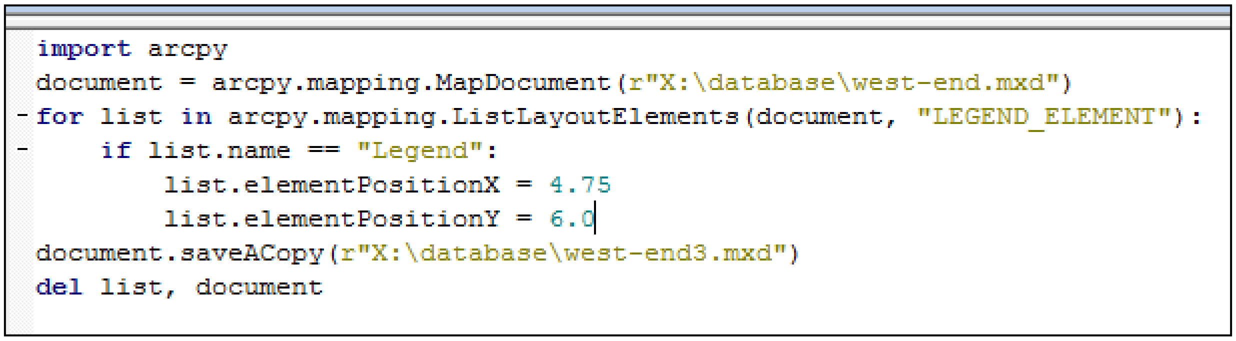


```
>>>  TEXT_ELEMENT
 TEXT_ELEMENT
 TEXT_ELEMENT
Alternating Scale Bar MAPSURROUND_ELEMENT
North Arrow MAPSURROUND_ELEMENT
Legend LEGEND_ELEMENT
Layers DATAFRAME_ELEMENT
```

Figure 14: The results of the Process

In Figure 24, the results are shown. Text elements (text boxes) have no name, but items such as the scale bar and directional arrow do have names and types. There is only one data frame called layers.

The layout of a map can be changed without having the map project file opened. While most people want to set-up the map elements according to the display needs, there are times such as placing specific elements in certain location that doing the operation outside of the mapping program may be advantageous. For example, say in a certain project a group of maps has been constructed, but the lead designer decided to move the company logo from the lower right corner to the upper right corner on all the maps. Fixing the layout outside of the mapping software might make a better usage of time, especially when multiple project (mxd) files might be part of what requires change.



```
import arcpy
document = arcpy.mapping.MapDocument(r"X:\database\west-end.mxd")
for list in arcpy.mapping.ListLayoutElements(document, "LEGEND_ELEMENT"):
    if list.name == "Legend":
        list.elementPositionX = 4.75
        list.elementPositionY = 6.0
document.saveACopy(r"X:\database\west-end3.mxd")
del list, document
```

Figure 15: Managing Layout

In Figure 25:

1. A variable document is defined to be the west-end project file.
2. A loop is used to look at the layout elements, which is looking specifically for the legend element.
3. Within the loop is an **if** statement, which looks specifically for the layout element named 'Legend',
4. When this statement is true the x and y position of the legend is changed. The project is saved with a new name and the variables are deleted.

See Figure 26 and 27, Figure 26 is the original location of the legend and Figure 27 is the new location.

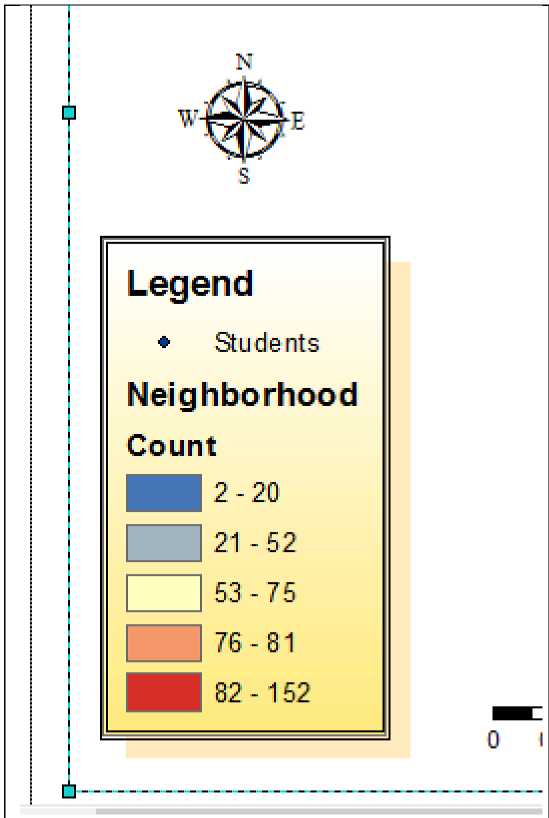


Figure 16: Original Position of the Legend

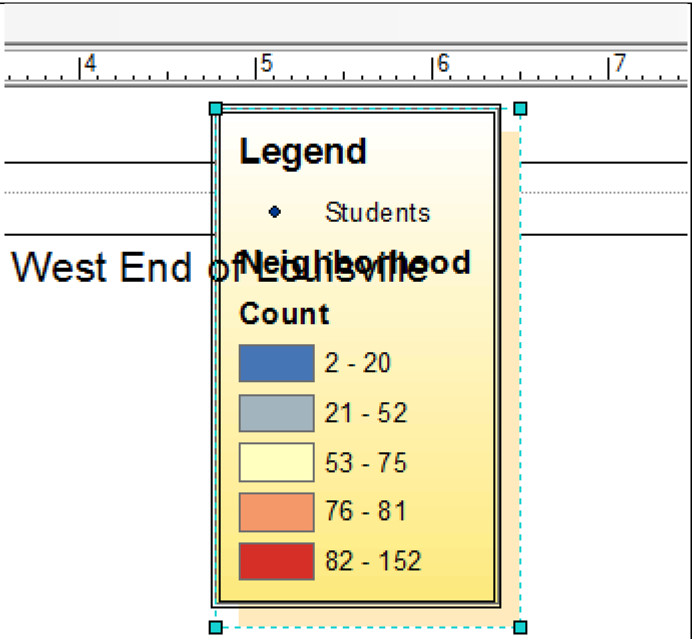


Figure 17: The Move Position of the Legend

Assignment 8.7

1. Determine the elements in a map project file. Make sure the project file has the normal mapping elements before the process begins.
2. Create a grid layout so that you can determine how the grid functions.
3. Create a formatted map from a script. This may take trial and error and make sure the intermediate map project files created are deleted or they could create problems if and overwrite is attempted.

Provide your instructor with the original map showing the grid and the layout of elements and a post-processing map showing how the elements were moved. Provide a script. The formatting of the final map does not have to follow good mapping conventions.