

EXERCISE 5

Geospatial Scripting in Python



Introduction

- This exercise focuses on using Python and ArcPy. This exercise is very fast paced, and builds on much of the information that you learned in exercises 1-3. It is designed to allow you to apply many of the concepts you learned in the earlier exercises and see how to use them in a geospatial scripting environment.

Objectives

- Apply and expand on the information that you learned in the exercises 1-3
- Learn more about Python and ArcPy

Required Software

- Python version 2.x

Required Data

- Course data folder

Prerequisites

- *Introduction to Geospatial Scripting Exercise 1: Planning a script.*
- *Introduction to Geospatial Scripting Exercise 2: Concepts of Scripting, Part 1*
- *Introduction to Geospatial Scripting Exercise 3: Concepts of Scripting, Part 2*
- Python installed



Table of Contents

| | |
|--|----|
| Part 1: Exploring ArcPy and Preparing a Script | 3 |
| Part 2: Loops | 6 |
| Part 3: Functions and Raster Math | 8 |
| Part 4: Notes and Other Resources | 11 |



Part 1: Exploring ArcPy and Preparing a Script

In this exercise you will see how to type code directly into a Python window in ArcMap, which is an interpreter like the Python shell. You will also prepare a standalone script in IDLE.

A. Opening a Python Window in ArcMap

1. **Open** ArcMap.
2. In ArcMap, open a Python window. **Click** Geoprocessing, then click Python. Or **Click** the Python window icon on the toolbar.



3. This opens a Python interpreter directly in in ArcMap. Each line you type into the interpreter is executed as you enter it into the window.
 - i. One advantage of this is the help window on the right. As you type in commands Python will give you information about the commands that you're using, such as which arguments you need to pass to a method.
 - ii. A disadvantage is that you cannot save these scripts. But to quickly use ArcMap tools, this is a nice environment to be in.

For this exercise, we will not be typing code into the ArcMap python window, instead we will be preparing a standalone script, later in the exercise. Keep ArcMap open for the next section.

B. Finding Syntax of ArcMap Tools

1. Open the Arc Toolbox. **Click** Geoprocessing, then ArcToolbox
2. In the window that opens, **Click** the plus sign to expand the Analysis toolbox, then **click** the plus sign to expand the Extract toolset.
3. **Right Click** the Clip tool. **Click** Item description.
4. In the Item Description window that opens there is information about the clip tool.
 - i. If you **scroll down** in the window you will see a section labeled Syntax, with the line:
`Clip_analysis (in_features, clip_features, out_feature_class, {cluster_tolerance})`.

Note: Let's break down what this instruction means piece by piece:

Clip_analysis (in_features, clip_features, out_feature_class, {cluster_tolerance})

"Clip_analysis" is the name of the tool. The item description is telling you what clip analysis will do, and how to use it.

Clip_analysis (in_features, clip_features, out_feature_class, {cluster_tolerance})

Everything in the parentheses, but not in the brackets, are the arguments that must be passed to the tool. These arguments are required for the tool to run. The item description will tell you what each of the arguments means in more detail.

Clip_analysis (in_features, clip_features, out_feature_class, {cluster_tolerance})

The arguments in brackets {} are optional arguments. If you provide nothing in these places, the tool will still run, you just won't be using that option. You don't need to include the brackets in your code, they're just there to indicate that the argument is optional.

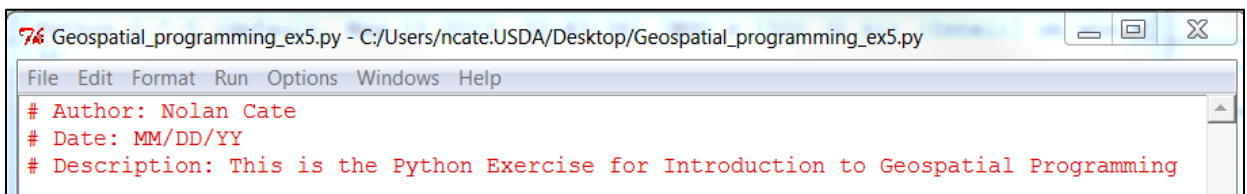
- ii. Continue **Scrolling down** on the page. The next thing you will see on the page is a table. The table describes the different arguments that can be passed to the tool (in the parameters column). It gives an explanation of what the argument does, and what data type in input or output.
- iii. Continue **Scrolling down**. There is a section labeled Code Samples. This section shows real examples of how to use the tool. Almost every tool in ArcMap will have a similar description page, so if you ever need guidance on how to use a tool this is an excellent resource.

C. Preparing a Standalone Script

1. To create a standalone script, **Open** an IDLE window.
2. In the IDLE window **Click** File , then New File (your version of Python could say "New Window" instead of "New File").
3. In the window that opens **Click** File, then Save as.
4. **Save the file** as Geospatial_Scripting_ex5.py. The .py extension is very important because it designated the file as a python program, which allows the python interpreter to use it.

Note: *Saving a standalone script will allow you to edit it and reuse it later. One huge advantage of writing code is that once you've written a useful tool, you shouldn't have to rewrite it. Code reusability is very important. As you begin to become a more advanced Python user you will begin to reuse old scripts, functions, and even write your own modules.*

5. As you learned in exercise 1, it is important to include a header in your script. On the first three lines **Add a Comment** with your name, the date, and a description of the script. Remember the # character will comment out a line in Python. Your script should now look similar to the example in figure 1.



```

7% Geospatial_programming_ex5.py - C:/Users/ncate.USDA/Desktop/Geospatial_programming_ex5.py
File Edit Format Run Options Windows Help
# Author: Nolan Cate
# Date: MM/DD/YY
# Description: This is the Python Exercise for Introduction to Geospatial Programming
    
```

Figure 1: Showing the header at the end of Part 1, Section C.

D. Importing Modules

Modules were mentioned briefly in Exercise 3, and they are very important to understand. See the note, below.

Note: A module is a code library. A code library allows programmers to develop and share commonly used pieces of code as a special script or library file that can be referenced and loaded into the main program. Before you import any modules, you have access to what is called the "Python Standard Library". This is all of the pieces of code, methods, operators, etc., that you have used so far in this training. Additional pieces of code exist in modules. These modules then extend the functionality of Python to include the imported code. These extra pieces of code in the modules are not available unless you specifically import the module.

For example, some of the math tools you used earlier are not part of the Python Standard Library, so they cannot be used when you first open Python. You have to import the module that they are a part of. For more information on modules and packages, see the resources section below or the glossary

Tools in ArcMap are written in a Python module called ArcPy, which you receive with ArcGIS. These tools aren't available normally in Python, so we have to tell the computer to import the ArcPy module so they will be.

1. It is best practice to begin your Python script by first writing the header, then importing all your modules. On the line below your header, type:

```
import arcpy
```

2. That's all there is to it. The arcpy module is now imported and tools from ArcMap can be used in your script. But ArcMap users will be aware that there are extensions in ArcMap that we may want to use that won't be available at first. We want to turn on Spatial Analyst. To do so, type:

```
arcpy.CheckOutExtension("Spatial")
```

Note: The . in this statement is an example of dot notation. In general, the dot tells the Python interpreter where to look for something. In this case, you are using the CheckOutExtension method and the dot is telling Python to find that method in the ArcPy module. You have seen some dot notation in previous exercises, and will continue to see dot notation used through the rest of the exercise.

E. Setting your Workspace Environment and Listing Rasters

In your standalone script, you will need to set your workspace environment. This going to be your working directory, where Python will look for geospatial files and where it will place files that you process.

1. Setting the workspace environment is a simple, single statement command. It is shown below.

```
arcpy.env.workspace = r"C:/GeospatialScripting"
```

What does this statement mean? arcpy is the site package where all the ArcGIS tools and class definitions are stored. The .env stands for environment, which is a class that you create an instance of every time you use ArcPy, and the .workspace is a property of that class instance, which we set equal to a path where the data for this exercise has been saved. Make sure to set the path to wherever you have saved your course data, this will become your working directory.

- Now that we've set a workspace, we can use arcpy tools here. One very useful tool is .ListRasters(). In the line below your workspace statement, add the code:

```
rasters = arcpy.ListRasters("L*", ".tif")
```

Now we have a variable equal to a list of rasters. In the ListRasters tool, we have passed two arguments. Both of these arguments are optional. The first is a "Wildcard", which we have told the tool to look for rasters beginning with an "L". The second argument is a file type, in this case .tif images.

Part 2: Loops

What is a loop? A loop is a piece of code that will execute repeatedly until some condition is met. Consider the following flowchart in figure 2:

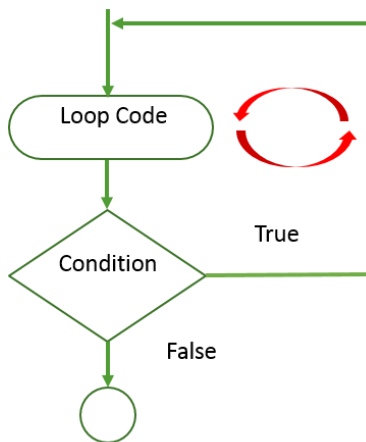


Figure 2: A flowchart showing the structure of a loop.

You can see that this flowchart looks similar to conditional statements that you learned about in earlier exercises. But in this case, the true condition runs some code, then returns the code to *before* the condition to test it again, so it will run as long as the condition is true. There are several different kinds of loops, their structures all differing slightly, here you're going to write a for loop.

A. For Loop

- On the next line of your script you're going to write a simple loop called a "for loop". This loop is going to print out each of the rasters in the raster list you created in the last section. The syntax of a for loop is very similar to the syntax of an if statement.

```
for image in rasters:
    print(image)
```

This means that the loop will cycle through the list rasters. Remember that the variable "rasters" is a list of the rasters in your workspace, which you created in the previous section. For each item in the list (in this loop we're calling the each item "image"), it will issue a print

statement. When there are no more items in the list, no more print statements can be issued and the loop will end.

2. Try running the code. **Click** Run, then Run Module. In the Python shell, a list of the rasters in your geoprocessing folder should have printed out. Your script right now should look similar to figure 3.

```
File Edit Format Run Options Windows Help
# Author: Nolan Cate
# Date: MM/DD/YY
# Description: This is the Python Exercise for Introduction to Geospatial Scripting

import arcpy
arcpy.CheckOutExtension("Spatial")

arcpy.env.workspace = r"C:/GeospatialScripting"

rasters = arcpy.ListRasters("L*", "TIF")

for image in rasters:
    print(image)
```

Figure 3: At this point in the exercise, your script should look similar to this.

3. Let's try using another useful set of arcpy functions inside of this for loop. **Delete** the print statement that you wrote in the first step of this section. **Create** a new variable called desc and set it equal to arcpy.Describe(image) – This creates a geoprocessing describe object.

```
desc = arcpy.Describe(image)
```

4. On the next line **Create** a new variable called pixelSize and set it equal to desc.meanCellHeight. desc is an object, a collection of data, that holds information about the rasters in our list. The method .meanCellHeight will extract the cell size (the spatial resolution) of the raster.

```
pixelSize = desc.meanCellHeight
```

Be aware of the dot notation here. Now the dot is telling Python to look for the property "meanCellHeight" in the object, desc.

5. Now **issue** a new print statement. This print statement is going to take a slightly different syntax than your previous print statements. See below:

```
print'The image {0} has a resolution of {1} meters'.format(image, pixelSize)
```

This print statements prints the string that is in quotes (the green text). The .format method fills in what should go in the numbers in the curly braces {}.

6. To see what these lines of code have done, **Click** Run, then Run Module. Save the script when you are prompted to do so. Your script should look similar to the example below:

```

File Edit Format Run Options Window Help
# Author: Nolan Cate
# Date: MM/DD/YY
#: Description: This is the Python Exercise for Introduction to Geospatial Scripting

import arcpy
arcpy.CheckOutExtension("Spatial")

arcpy.env.workspace = r"C:/GeospatialScripting"

rasters = arcpy.ListRasters("L*", "TIF")

for image in rasters:
    print(image)
    desc = arcpy.Describe(image)
    pixelSize = desc.meanCellHeight
    print'The image {0} has a resolution of {1} meters'.format(image,pixelSize)
    
```

Part 3: Functions and Raster Math

You learned about functions in exercise 3. Now you're going to get to create a useful function in your code and see how to call it. This function is going to create a Normalized Difference Vegetation Index (NDVI) image.

A. Setting up the Function

1. Recall from exercise 3 that to **define** a function you need to specify the name of the function, and the arguments that must be passed to it. We're going to name this function `runNDVI`, and the arguments it needs are a Near Infrared image and a Red image. **Type** the following code below your for loop:

```
def runNDVI(nir, red):
```

2. Like the for loop, everything that is indented after the colon is part of the function. **Hit** enter to go the next line. IDLE should automatically indent the code for you.

B. Raster Math in the Body of the Function

1. On the first line in the function you're going to calculate the numerator of the NDVI equation. **Create** a local variable called `ndviNum`. Remember that all variables created in the function will be local variables, which means they can't be used outside of the function they're created in. You'll want this variable to be an image of floating point values equal to the nir image minus the red image. The syntax for doing to is below:

```
ndviNum = arcpy.sa.Float(nir-red)
```

2. **Hit** enter to go the next line. IDLE recognizes Python syntax and will automatically indent your code.

3. On the second line of the function you want to calculate the denominator of the NDVI equation. **Create** a local variable called `ndviDenom`. You'll want this to be an image of floating point values equal to the nir image plus the red image. The syntax for doing to is below:

```
ndviDenom = arcpy.sa.Float(nir+red)
```

4. **Hit** enter to go the next line. IDLE should still be automatically indenting the code for you.

5. To create the vegetation index, just complete the NDVI equation. **Create** a new local variable called `outRaster` and set it equal to the numerator over the denominator. See the syntax below:

```
outRaster = ndviNum/ndviDenom
```

6. **Hit** enter to go the next line. IDLE should still be automatically indenting the code for you.

7. Now all you need to do is tell the function what to return when it's called later in the script. You want the returned image to be the index variable. So **type** `return index`. The complete function should look like this:

```
def runNDVI(nir, red):
    ndviNum = arcpy.sa.Float(nir-red)
    ndviDenom = arcpy.sa.Float(nir+red)
    outRaster = ndviNum/ndviDenom
    return outRaster
```

8. **Hit** enter to go the next line. After the return statement, IDLE should no longer automatically indent the code for you. If it does, type backspace to remove the indent. At this point your script should look like this:

```
File Edit Format Run Options Window Help
# Author: Nolan Cate
# Date: MM/DD/YY
#: Description: This is the Python Exercise for Introduction to Geospatial Scripting

import arcpy
arcpy.CheckOutExtension("Spatial")

arcpy.env.workspace = r"C:/GeospatialScripting"

rasters = arcpy.ListRasters("L*", "TIF")

for image in rasters:
    print(image)
    desc = arcpy.Describe(image)
    pixelSize = desc.meanCellHeight
    print('The image {0} has a resolution of {1} meters'.format(image,pixelSize))

def runNDVI(nir, red):
    ndviNum = arcpy.sa.Float(nir-red)
    ndviDenom = arcpy.sa.Float(nir+red)
    outRaster = ndviNum/ndviDenom
    return outRaster
```

C. Calling the Function

Calling a function is a relatively simple statement, but it is important to understand all of the pieces of the statement. The following steps will guide you through calling the function in a single line.

1. The function `runNDVI` will calculate an NDVI image from a NIR and red image that you pass to it. To call this function, **create** a new global variable called `ndvi`, and set it equal to the function `runNDVI`. Remember that a global variable is any variable that is created outside of a function and it can be used anywhere else in the script.
2. In the parentheses after the function you need to specify the arguments. Remember when you defined this function in the last section you set the necessary arguments to a NIR image, and a red image so the function will be expecting two images and it will use them in that



order. First pass the NIR image. This is band 5 of the Landsat image you are working with, and all of these bands are listed in the "rasters" variable that you created earlier. To point the computer towards this particular raster, **type** `arcpy.sa.Raster(rasters[6])`. So far your single line statement should look like this:

```
ndvi = runNDVI(arcpy.sa.Raster(rasters[6]))
```

3. So far this statement has called the runNDVI function and passed it the first argument, the NIR image. Now we must pass it the red image. The syntax for this is the same for the NIR image, but you need to separate the two with a comma, and change the list index (the number inside the square brackets) to 5. Your complete statement should look like this:

```
ndvi = runNDVI(arcpy.sa.Raster(rasters[6]), arcpy.sa.Raster(rasters[5]))
```

4. Now the ndvi variable is equal to the output of the runNDVI function, when you pass it a Near Infrared band and a red band.

D. Saving the Raster

1. Now that the vegetation index had been generated you need to save it to your computer so you can access it later. The raster class has a save method that we can use to get the image onto our machine. The argument passed to this method is an output path. Apply the save method to the ndvi variable.

```
ndvi.save(r"C:\GeospatialScripting\ndviOut.tif")
```

2. **Add** another print statement after saving the raster to indicate that the script is finished running. Print the string, 'done'

```
print 'done'
```

This print statement will execute after the raster has finished saving, so when we run the script it tell us when it is finished.

3. **Run the Script**

4. The print statements supplying the resolution of the images should appear first. The ndvi will take a moment to process. When the statement, 'done', appears in the shell, the script is finished.

```
File Edit Format Run Options Window Help
# Author: Nolan Cate
# Date: MM/DD/YY
#: Description: This is the Python Exercise for Introduction to Geospatial Scripting

import arcpy
arcpy.CheckOutExtension("Spatial")

arcpy.env.workspace = r"C:/GeospatialScripting"

rasters = arcpy.ListRasters("L*", "TIF")


for image in rasters:
    print(image)
    desc = arcpy.Describe(image)
    pixelSize = desc.meanCellHeight
    print('The image {0} has a resolution of {1} meters'.format(image,pixelSize))

def runNDVI(nir, red):
    ndviNum = arcpy.sa.Float(nir-red)
    ndviDenom = arcpy.sa.Float(nir+red)
    outRaster = ndviNum/ndviDenom
    return outRaster

ndvi = runNDVI(arcpy.sa.Raster(rasters[6]),arcpy.sa.Raster(rasters[5]))
ndvi.save(r"C:\GeospatialScripting\ndviOut.tif")

print'done'
```

E. Viewing the Raster in ArcMap

1. **Open** an ArcMap window.
2. **Click** the add data button.
 
3. In the add data window, **navigate** to where you saved the ndviOut raster.
4. **Open** the ndviOut.tif raster. ArcMap will display the NDVI raster that you generated using the script that you just wrote. Values will be between -1 and 1, with higher values highlighting healthier vegetation.

Part 4: Notes and Other Resources

- GTAC offers additional trainings in Python. The tutorial of the course *Introduction to Python for Geoprocessing* is available here: http://fsweb.geotraining.fs.fed.us/www/index.php?view_unit=4867&lessons_ID=3040
- ESRI provides significant documentation about arcpy syntax both in ArcMap and in online documentation. A link to some online resources is here: <http://pro.arcgis.com/en/pro-app/arcpy/get-started/what-is-arcpy-.htm>
- The internet will prove to be an invaluable resource. If you're having any trouble with a script, chances are somebody else will have encountered an identical problem, found a solution, and provided it online. Often these solutions can be found on GIS forums. One such useful forum is: <http://gis.stackexchange.com/>



- Python is more than arcpy. The Python portion of this course focused on using ArcPy for geospatial processing. But Python can be used for so much more than just the tools in the ArcPy site package. There are many geospatial specific tools available to use. See this link for a sample of a few:

http://www.data-analysis-in-python.org/t_gis.html

Congratulations! You have learned some basic and very important skills about geospatial scripting in Python. This should have given you the chance to practice some of the skills you learned in earlier exercises and given you some resources to continue learning. Remember that when scripting you are learning a new language, and it will take time. Review this exercise and visit the additional resources provided to keep practicing your new skills.

