EXERCISE 7
Geospatial Scripting in Python 3

Introduction

This exercise focuses on using Python 3 and Jupyter Notebook in ArcPro. 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

- Learn more about Python and JupyterNotebook

Required Data

- Python Course data folder

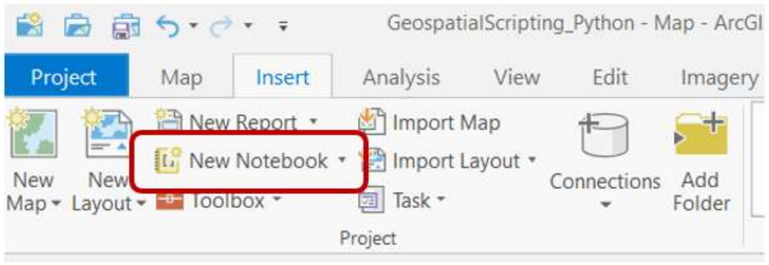
Table of Contents

Part 1: Exploring ArcPy and Preparing a Script

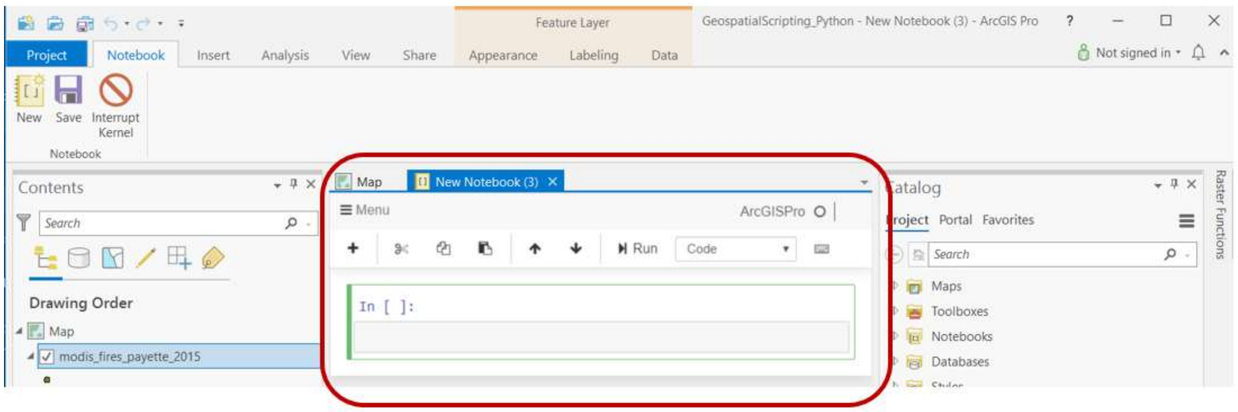
In this exercise you will see how to type code directly into a Python window in ArcPro, which is an interpreter like the Python shell. You will also prepare a standalone script in IDLE.

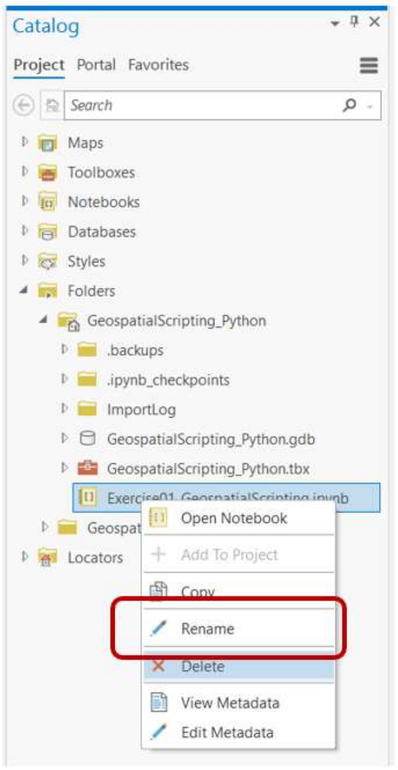
A. Create a Jupyter Notebook script

1. Open a Jupyter Notebook in ArcPro by going to Insert>New Notebook



It will automatically be named New Notebook





Rename the notebook by navigating to the **Catalog> Folder>GeospatialScripting** the notebook should be saved in the project geodatabase right-click and select **Rename**

Rename the notebook **GeospatialScripting_Ex7**

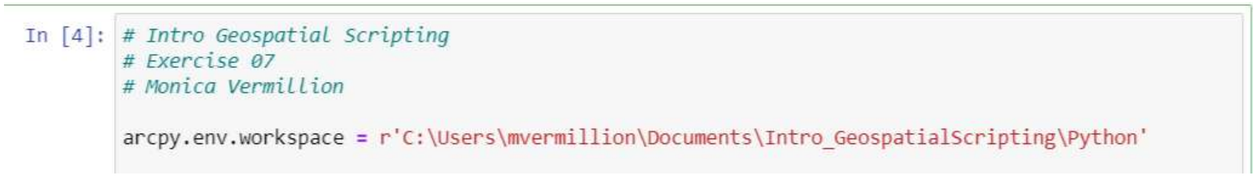
2. Set the workspace by replacing USERNAME with your username and setting the path as the path to your saved course data

```
arcpy.env.workspace = r'C:\Users\USERNAME\Documents\GeospatialScripting\Course Data'
```

3. Check out the spatial analyst extension

```
arcpy.CheckOutExtension("spatial")
```

4. Add a comment to the top of the script that includes your name, the name of the exercise. Then start to add comments to each line of the script to describe what it does.

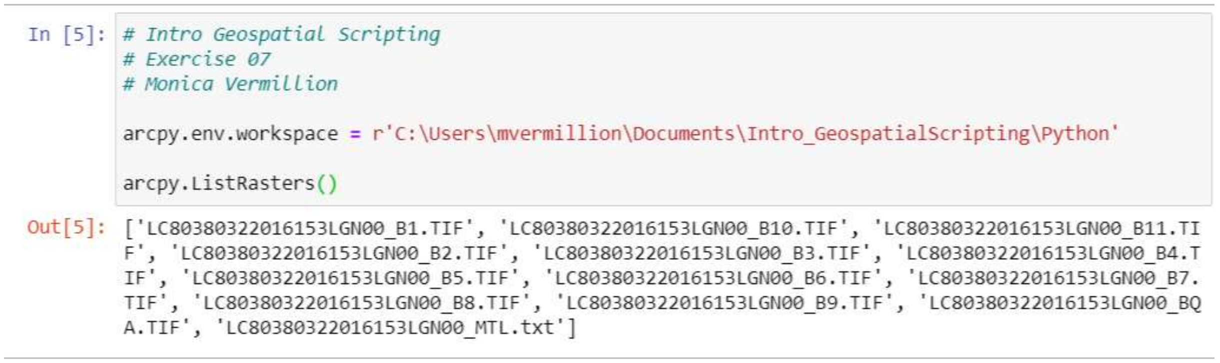


```
In [4]: # Intro Geospatial Scripting
        # Exercise 07
        # Monica Vermillion

        arcpy.env.workspace = r'C:\Users\mvermillion\Documents\Intro_GeospatialScripting\Python'
```

5. Add the `ListRasters` command, start by typing **arcpy.List** notice in Jupyter Notebooks it does not autocomplete, you must press the **Tab** button to see the autocomplete options

```
arcpy.ListRasters()
```

Run the script, it should look like the script below:



```
In [5]: # Intro Geospatial Scripting
        # Exercise 07
        # Monica Vermillion

        arcpy.env.workspace = r'C:\Users\mvermillion\Documents\Intro_GeospatialScripting\Python'

        arcpy.ListRasters()

Out[5]: ['LC80380322016153LGN00_B1.TIF', 'LC80380322016153LGN00_B10.TIF', 'LC80380322016153LGN00_B11.TI
        F', 'LC80380322016153LGN00_B2.TIF', 'LC80380322016153LGN00_B3.TIF', 'LC80380322016153LGN00_B4.T
        IF', 'LC80380322016153LGN00_B5.TIF', 'LC80380322016153LGN00_B6.TIF', 'LC80380322016153LGN00_B7.
        TIF', 'LC80380322016153LGN00_B8.TIF', 'LC80380322016153LGN00_B9.TIF', 'LC80380322016153LGN00_BQ
        A.TIF', 'LC80380322016153LGN00_MTL.txt']
```

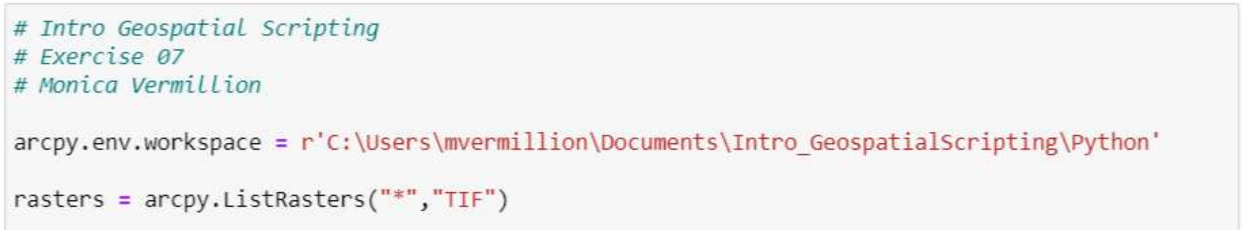
6. We only want the rasters that are .TIF rasters, we don't want the .txt file, to do this we will use the arguments to the **.ListRasters()** method:

```
arcpy.ListRasters(“*”,”TIF”)
```

7. Cast the list of rasters to the variable **rasters**

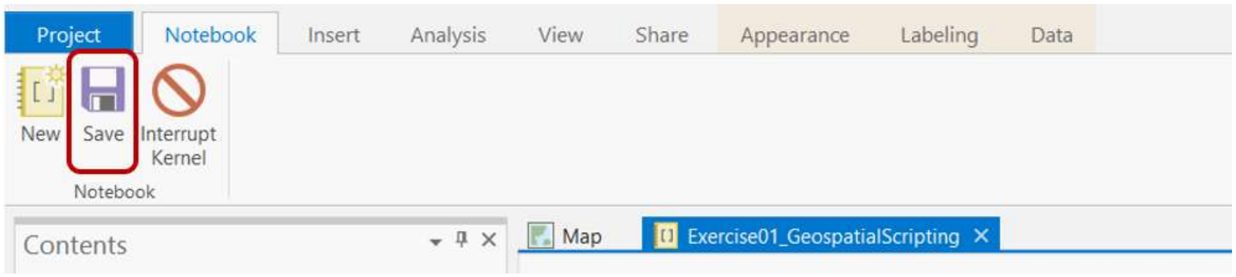
```
rasters = arcpy.ListRasters(“*”,”TIF”)
```

To run the code either press ctrl+ enter or hit the ⏯ Run button at the top of the notebook to run the script



```
# Intro Geospatial Scripting
# Exercise 07
# Monica Vermillion

arcpy.env.workspace = r'C:\Users\mvermillion\Documents\Intro_GeospatialScripting\Python'

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

8. Save the script using the Save button in the upper left-hand corner

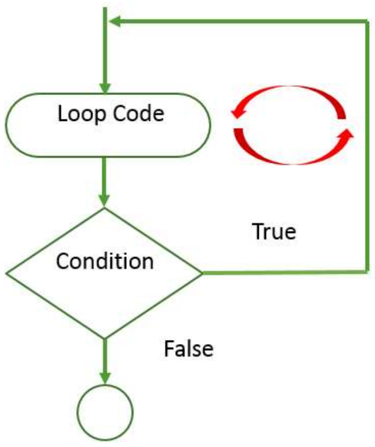


When to use a Jupyter Notebook vs. the Python Window:

For the most part you should save your work and use Jupyter Notebooks, however when testing a bit of code troubleshooting the Python Window is a great option.

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:



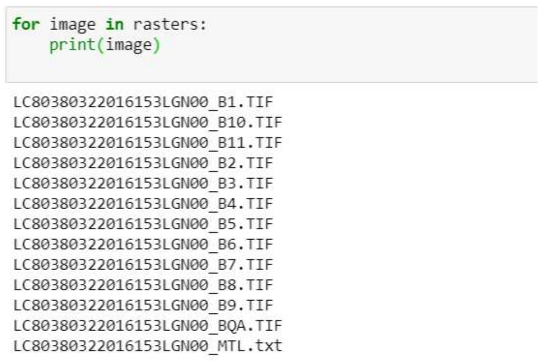
You can see that this flowchart looks like conditional statements that you learned about in earlier exercises. But in this case, the true condition runs some code, then returns the code to *before* the condition to test it again, so it will run if the condition is true. There are several different kinds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

A. For Loop

1. On the next line of your script, you’re going to write a "for loop". This loop is going to print out each of the raster names 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)
```

Don’t forget a comment to explain what the code does! 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. For each item in the list (in this loop we're calling the each item "image") it will issue a print statement. Try running the code:



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

```
LC80380322016153LGN00_B1.TIF
LC80380322016153LGN00_B10.TIF
LC80380322016153LGN00_B11.TIF
LC80380322016153LGN00_B2.TIF
LC80380322016153LGN00_B3.TIF
LC80380322016153LGN00_B4.TIF
LC80380322016153LGN00_B5.TIF
LC80380322016153LGN00_B6.TIF
LC80380322016153LGN00_B7.TIF
LC80380322016153LGN00_B8.TIF
LC80380322016153LGN00_B9.TIF
LC80380322016153LGN00_BQA.TIF
LC80380322016153LGN00_MTL.txt
```

2. 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)
```

3. 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.

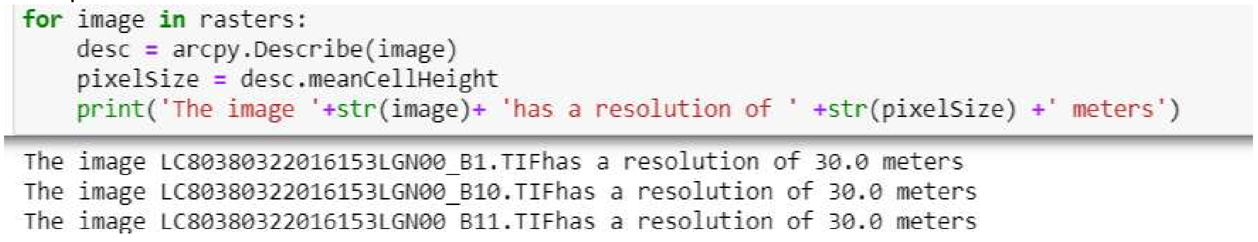
```
pixelSize = desc.meanCellHeight
```

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

4. 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 '+str(image)+ 'has a resolution of ' +str(pixelSize) +' meters')
```

5. To see what these lines of code have done, click **Run**. Your script should look similar to the example below:



```
for image in rasters:
    desc = arcpy.Describe(image)
    pixelSize = desc.meanCellHeight
    print('The image '+str(image)+ 'has a resolution of ' +str(pixelSize) +' meters')
```

```
The image LC80380322016153LGN00_B1.TIFhas a resolution of 30.0 meters
The image LC80380322016153LGN00_B10.TIFhas a resolution of 30.0 meters
The image LC80380322016153LGN00_B11.TIFhas a resolution of 30.0 meters
```

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 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 the for loop:

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

   Like the for loop, everything that is indented after the colon is part of the function

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**, for NDVI Numerator. 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 a raster 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)
```

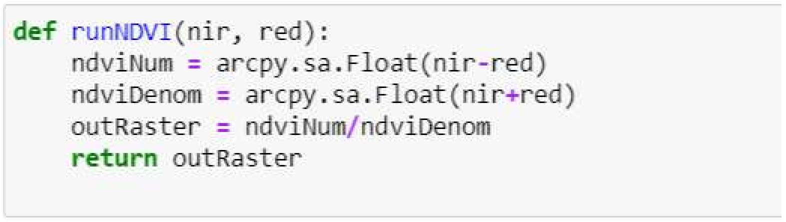
2. On the second line of the function you want to calculate the denominator of the NDVI equation. Create a local variable called **ndviDenom**. You'll want this to be an image of floating-point values equal to the nir image plus the red image. The syntax for doing to is below:

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

3. 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:

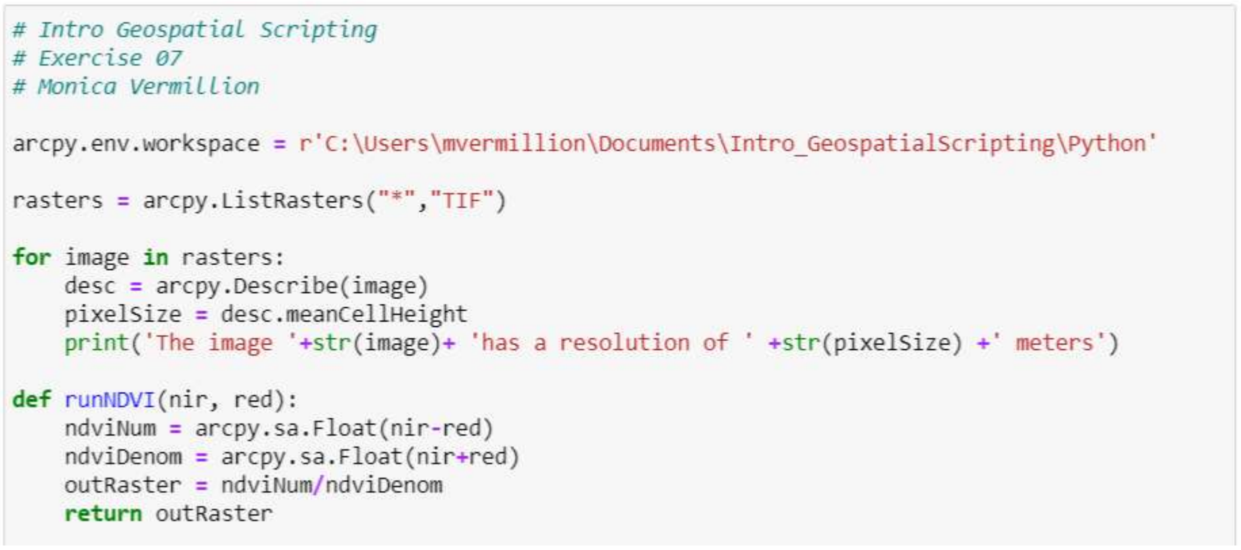
```
outRaster = ndviNum/ndviDenom
```

4. Now all you need to 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
```

5. At this point your script should look like this:



```
# Intro Geospatial Scripting
# Exercise 07
# Monica Vermillion

arcpy.env.workspace = r'C:\Users\mvermillion\Documents\Intro_GeospatialScripting\Python'

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

for image in rasters:
    desc = arcpy.Describe(image)
    pixelSize = desc.meanCellHeight
    print('The image '+str(image)+ 'has a resolution of ' +str(pixelSize) +' meters')

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

The following steps will guide you through calling the function:

1. Create the arguments: **Landsat_red** and **Landsat_nir**
   i. Create variable red. This is band 4 of the Landsat image you are working with, and all these bands are listed in the "rasters" variable that you created earlier. To point the computer towards this raster we will index the **rasters** variable

```
Landsat_red = arcpy.sa.Raster(rasters[5])
print(Landsat_red)
```

ii. Create the variable **nir** This is band 5 of the Landsat image you are working with, and all these bands are listed in the "rasters" variable that you created earlier. To point the computer towards this raster we will index the **rasters** variable

```
Landsat_nir = arcpy.sa.Raster(rasters[6])
print(Landsat_nir)
```

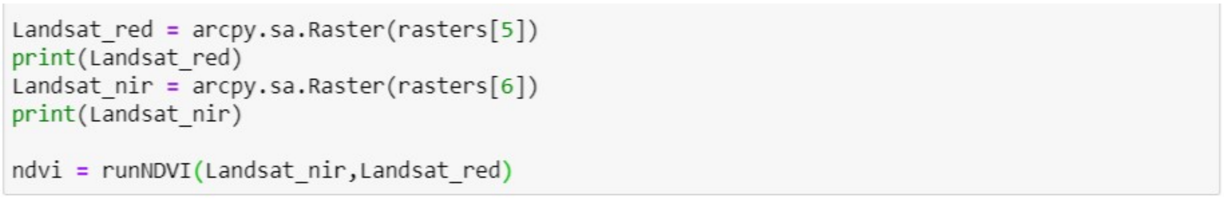
2. The function **runNDVI** will calculate an NDVI image from a NIR and red image that you pass to it. To call this function, create a new global variable called **ndvi**, and set it equal to the function **runNDVI**. Remember that a global variable is any variable that is created outside of a function and it can be used anywhere else in the script.

```
ndvi = runNDVI(
```

3. 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, **nir** and a red image, **red** so the function will be expecting two images and it will use them in that order. Your statement should look like this:

```
ndvi = runNDVI(Landsat_nir,Landsat_red)
```

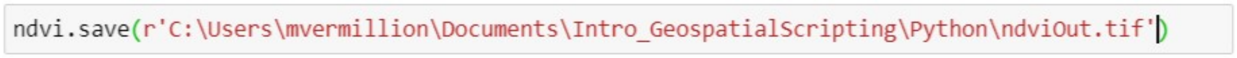
4. Now the **ndvi** variable is equal to the output of the runNDVI function, when you pass it a Near Infrared band and a red band.



```
Landsat_red = arcpy.sa.Raster(rasters[5])
print(Landsat_red)
Landsat_nir = arcpy.sa.Raster(rasters[6])
print(Landsat_nir)

ndvi = runNDVI(Landsat_nir,Landsat_red)
```

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** object 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, and change the path with your path to your output:



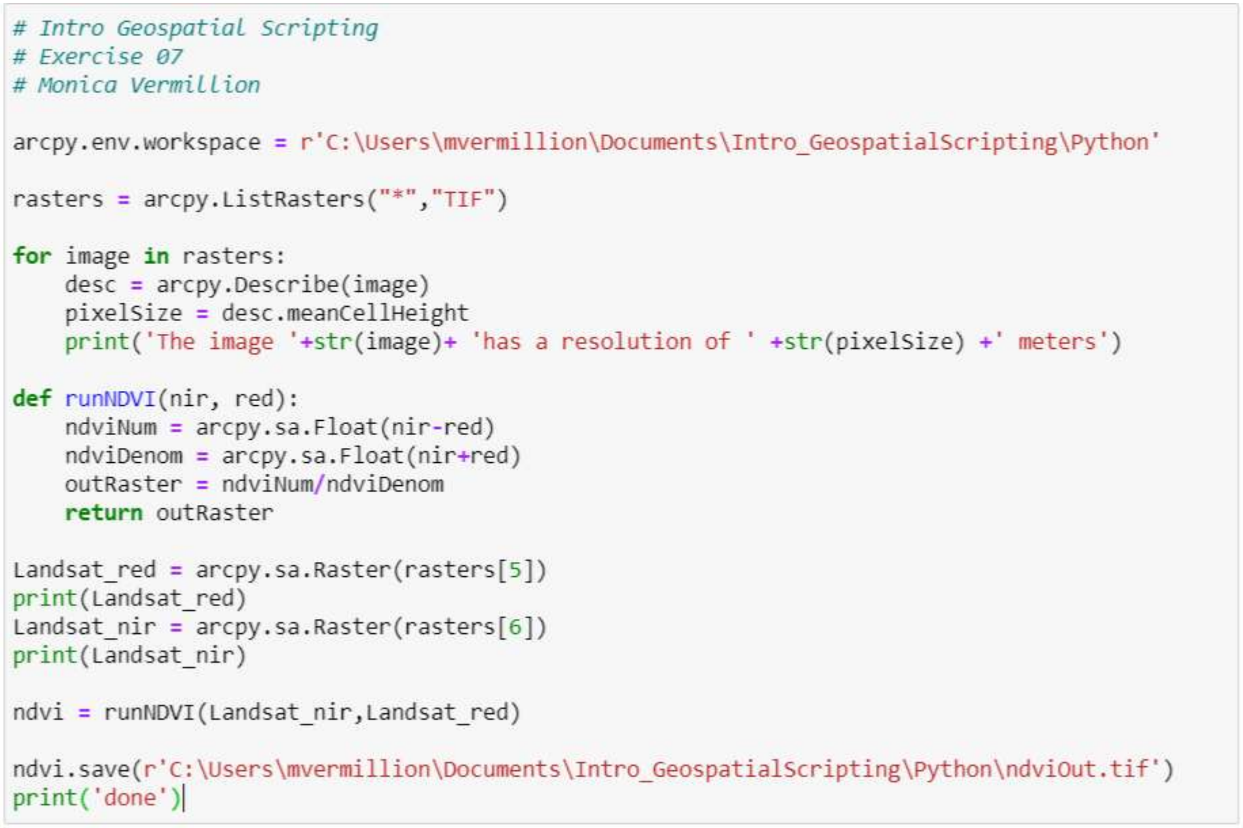
```
ndvi.save(r'C:\Users\mvermillion\Documents\Intro_GeospatialScripting\Python\ndviOut.tif')
```

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. **Run and Save the Script**

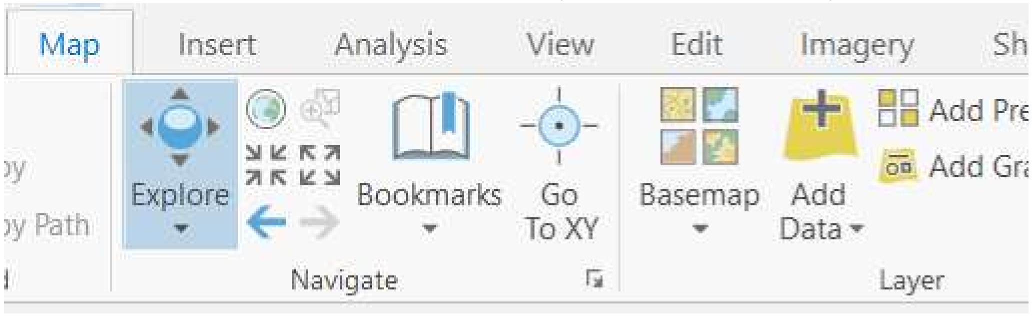
3. The print statements supplying the resolution of the images should appear first. The ndvi will take a moment to process. When the statement, 'done', appears in the shell, the script is finished.



```
# Intro Geospatial Scripting
# Exercise 07
# Monica Vermillion

arcpy.env.workspace = r'C:\Users\mvermillion\Documents\Intro_GeospatialScripting\Python'

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

for image in rasters:
    desc = arcpy.Describe(image)
    pixelSize = desc.meanCellHeight
    print('The image '+str(image)+ 'has a resolution of ' +str(pixelSize) +' meters')

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

Landsat_red = arcpy.sa.Raster(rasters[5])
print(Landsat_red)
Landsat_nir = arcpy.sa.Raster(rasters[6])
print(Landsat_nir)

ndvi = runNDVI(Landsat_nir,Landsat_red)

ndvi.save(r'C:\Users\mvermillion\Documents\Intro_GeospatialScripting\Python\ndviOut.tif')
print('done')
```

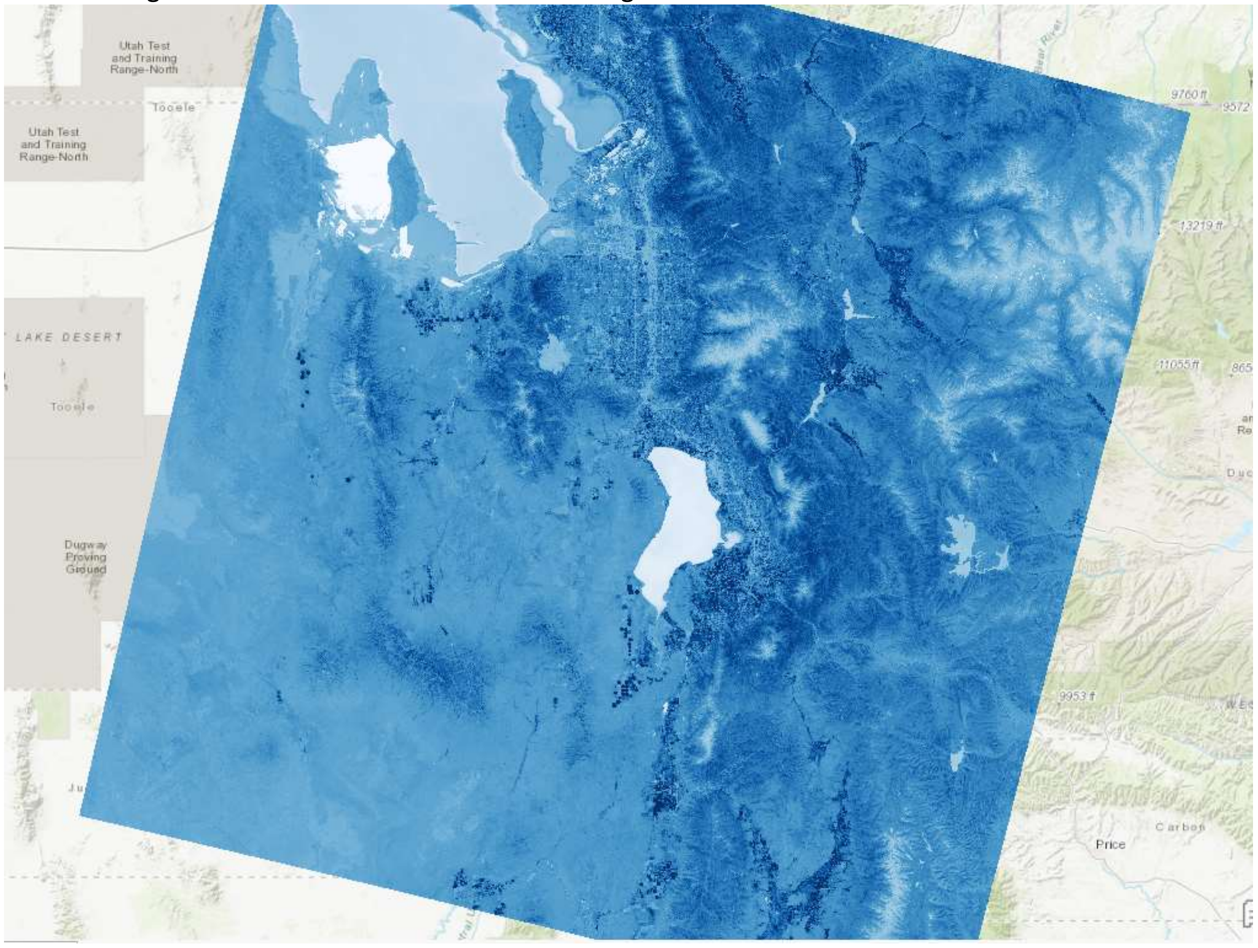
E. Viewing the Raster in ArcPro

1. Click on the Map tab and then in the top ribbon select the Map tab



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. ArcPro 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. Click on the colorbar and change the color scheme to visualize the data



Part 4: Notes and Other Resources

- ESRI provides significant documentation about arcpy syntax both in ArcMap and in online documentation. A link to some online resources is here
- 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.
- 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: Python Data Science Handbook

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.