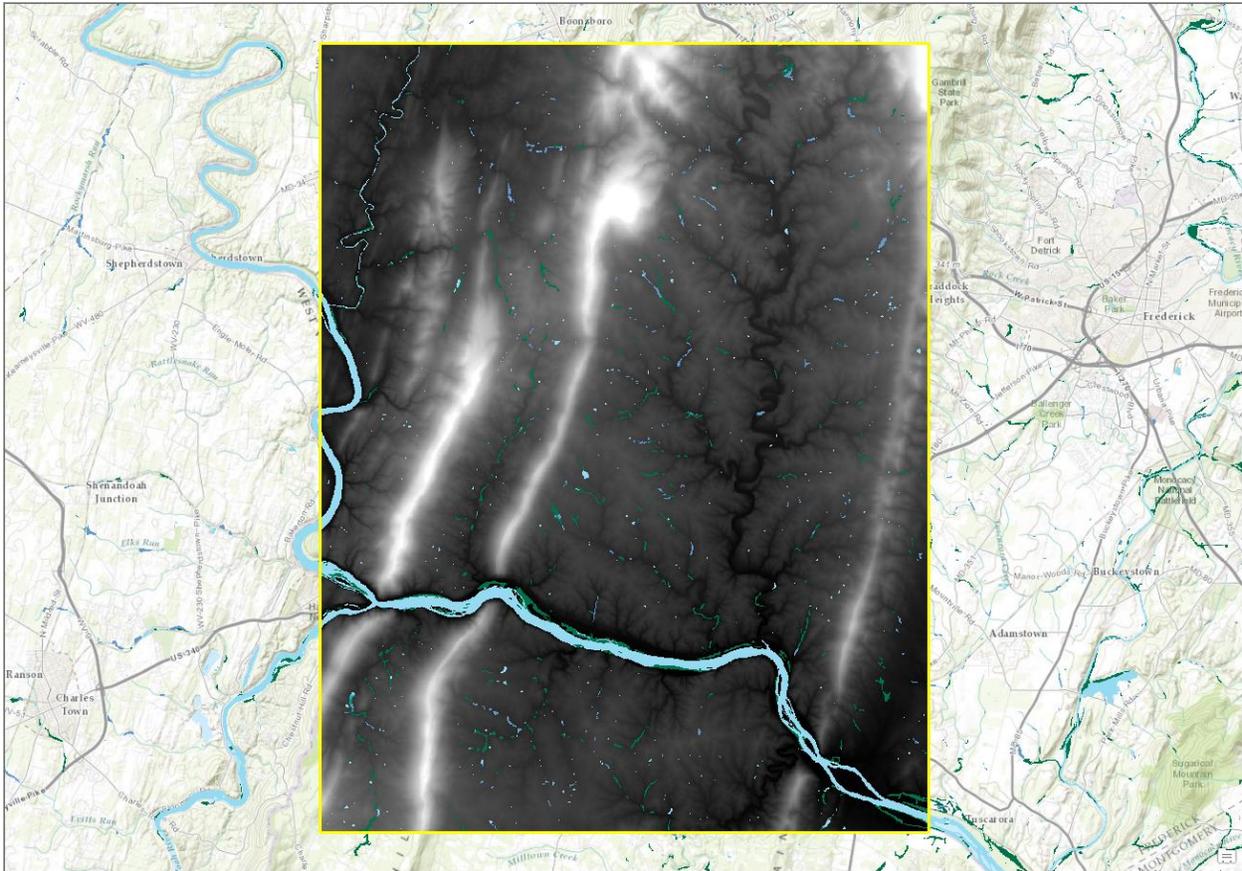


Introduction to Python

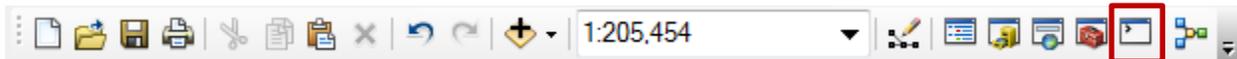
Through the course of these exercises we are going to create a Python script that identifies steep slopes close to wetlands and buffers them to create a minimal disturbance area.

Exercise 1: Getting Started

- 1) Open up the WetlandAnalysis.mxd document in ArcMap. You'll see 3 layers:
 - a. USGS National Elevation Dataset information for our Area of Interest
 - b. National Wetlands Inventory information
 - c. The Area of Interest



- 2) To get started using Python, let's work with the interactive Python command line available in ArcMap. From the Standard toolbar, select the Python window button:

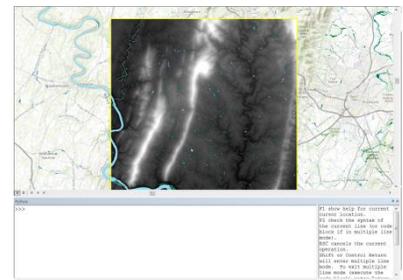


This will open up the Python window; I find it convenient to place it in the bottom of the application, below the map (as seen in the image to the right).

- 3) To get started, let's try using some simple commands. In the Python Window, type the following and then press Enter/Return:

2 + 2

You should see the number 4 appear in grey text below the line and a new line appear. The grey text is the response from command – in this case the evaluation of adding the numbers together



- 4) Now let's create a command that assigns a number to a variable. Variables hold values that can be re-used later in a process. Type the following in and press Enter:

```
myNumber = 2 + 2 * 4
```

You won't see a response from the Python window, just a new line after entering in the command. We just created a variable called myNumber and given it a value. In this case the value is 10, because multiplication is evaluated before addition. Verify the number is 10 by typing in myNumber in the command line (you should notice that the Python Window will want to autocomplete the variable name; you can select it with arrow keys and have it fill in using the Tab key). If we had wanted the addition to be evaluated first, we would have entered (2 + 2) * 4 – anything inside parenthesis is always evaluated first.

- 5) Another type of data we use in programming is text, which we call 'strings'. Create a variable called myName with your first name by typing it inside double quotation marks (i.e., "James"). Strings can be thought of as a series of letters. Run the command

```
len(myName)
```

You get back how many letters long the name is. Notice that when you type in len, the help box on the right describes the command and what it requires to run. Typing in

```
myName[0]
```

Provides the first letter of the name.

- 6) Joining strings together is called concatenation, and it also uses the + symbol. Try typing in

```
"Hello " + myName
```

The output is the text joined together. Now try

```
myName + "'s number is " + myNumber (that's a double quote followed by a single quote)
```

You get an error- you can't join together strings and numbers in Python. To make that work, we need to convert what myNumber is holding into a string; fortunately, the command is very easy – str(myNumber) does the conversion:

```
myName + "'s number is " + str(myNumber)
```

- 7) So far, we've been using the built-in functionality of Python. There are many times (including accessing ArcGIS commands) that we want to be using functionality not built in. To do that, we use a command called import to load the functionality (assuming it's been installed). As an example, type in the following:

```
import random
```

The random module contains functions for generating random numbers, which is useful in simulations and sample design. Let's update the myNumber variable with a random number between 0 and 100:

```
myNumber = random.randrange(0,100)
```

Now if you type in myNumber, you'll see a new value (unless you randomly picked 10!).

- 8) Let's use your new number to highlight how if statements work. Type in the following and press Enter:

```
if myNumber >= 50:
```

Notice that next line comes up, indented under the first line. The colon indicates we're creating a block, a set of commands that will run at once (in this case if myNumber is greater than or equal to 50). On the next line, type the following:

```
    print "The number is big"
```

Then press return. The same indented line comes up. In this case, we've finished the block for when the if statement is true; we need to put in what happens when it's false. Backspace out the 4 characters you're indented in and type the following

```
else:
```

```
    print "The number is small"
```

In all, it should look like:

```
if myNumber >= 50:  
    print "The number is big"  
else:  
    print "The number is small"
```

Press Enter after the final line to have the entire set of lines run. Is your number big or small?

- 9) Another type of block command we commonly use is a loop, where we do the same set of commands over a set of changing values. Type in the following and press enter on a blank line when done:

```
countdown = 0  
while myNumber >= countdown:  
    print myNumber - countdown  
    countdown = countdown + 1
```

You'll see a series of numbers appear on the screen, decreasing from the value of your number.

- 10) To finish this introductory exercise, let's run an ArcGIS tool. The arcpy module is already loaded; type in the following:

```
arcpy.env.workspace
```

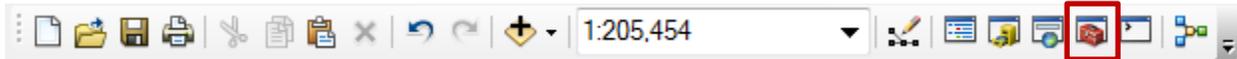
This tells us the workspace (default storage location) Python is working with. To run a command, we use the pattern `arcpy.<Command Name>_<Toolbox>`. As an example, type in the following to clip the wetlands to the footprint and press Enter:

```
arcpy.Clip_analysis('Wetlands', "Area of Interest", "Wetlands_Clipped")
```

Note that the layer names will also autofill as you get to that part in the command. This makes it easier to work with the map document.

Exercise 2: Model to Script

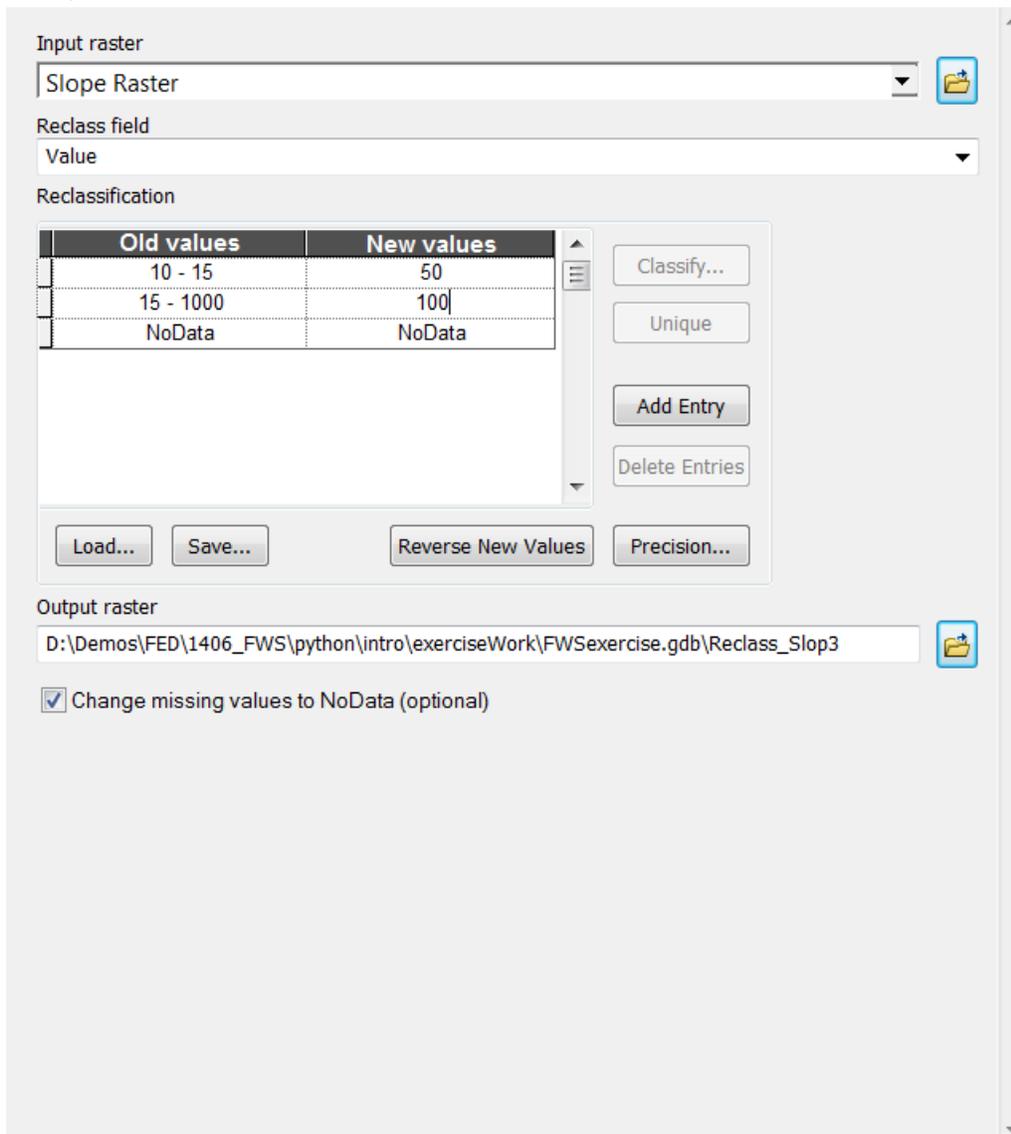
- 1) We're going to create a model to identify steep slopes and make the steep slope areas into polygons. Find the 'Toolbox' in the workspace, and edit the existing, blank Model by right-clicking on it and select 'Edit'.
- 2) To generate the slope information, we need to begin with the elevation data, so drag the layer from the Table of Contents into the ModelBuilder window.
- 3) The next tools we're going to use require the Spatial Analyst extension. Go to the 'Customize' menu and select 'Extensions'. Make sure 'Spatial Analyst' is checked on and press OK.
- 4) Now open up ArcToolbox using the button on the toolbar:



- 5) The Slope command is in the 'Spatial Analyst' Toolbox, inside the 'Surface' Toolset. Select it and drag it into ModelBuilder. The command comes in uncolored because we need to define input to have it run successfully.
- 6) Connect the Elevation Oval to the Slope Box in ModelBuilder by selecting the connection tool and then clicking on the objects in order.



- 7) Next we need to isolate the steep slopes. We can do that using a tool called Reclassify (in the Reclass toolset), which will change the values of a raster. Drag the tool into ModelBuilder and connect the output of the slope tool to it. Double-click on the tool and fill it in as you see below (ignore the differences in 'Input Raster') and then press 'OK':

A screenshot of the Reclassify tool dialog box in ArcGIS. The dialog box has a light gray background and contains the following elements:

- Input raster:** A dropdown menu showing 'Slope Raster' with a folder icon to its right.
- Reclass field:** A dropdown menu showing 'Value' with a downward arrow.
- Reclassification:** A table with two columns: 'Old values' and 'New values'.

Old values	New values
10 - 15	50
15 - 1000	100
NoData	NoData
- Buttons:** 'Classify...', 'Unique', 'Add Entry', 'Delete Entries', 'Load...', 'Save...', 'Reverse New Values', and 'Precision...'.
- Output raster:** A text field containing the path 'D:\Demos\FED\1406_FWS\python\intro\exerciseWork\FWSexercise.gdb\Reclass_Slop3' with a folder icon to its right.
- Checkboxes:** A checked checkbox labeled 'Change missing values to NoData (optional)'.

- 8) After generating the steep slopes, we'll need to work with them as polygons. From the Conversion Toolbox, open the 'From Raster' toolset and select the 'Raster to Polygon' tool. Connect the output of the Reclassify tool to the Raster to Polygon tool. Also, right-click on the 'Raster to Polygon' tool's output and select the entry 'Model Parameter'. When done, your model should look something like this:



- 9) Save the Model by clicking on the 'Save' button'.
 10) Now let's export this model into a script. Go to the 'Model' menu and select 'Export > To Python Script' option.
 11) Let's look at the script. In Windows, open the workspace folder. Right-click on the createSteepSlopes.py file (the .py might be missing) and select 'Edit in IDLE'. This will open up the file in IDLE, a Python editing program. It should somewhat like this:

```

# -*- coding: utf-8 -*-
# -----
# createSteepSlopes.py
# Created on: 2014-06-15 22:14:30.00000
# (generated by ArcGIS/ModelBuilder)
# Usage: createSteepSlopes <Steep_Slope_Polygons>
# Description:
# -----

# Import arcpy module
import arcpy

# Check out any necessary licenses
arcpy.CheckOutExtension("spatial")

# Script arguments
Steep_Slope_Polygons = arcpy.GetParameterAsText(0)
if Steep_Slope_Polygons == '# ' or not Steep_Slope_Polygons:
    Steep_Slope_Polygons = "D:\Demos\FED\1406_FWS\python\intro\exerciseWork\FWSexercise.gdb\RasterT_Reclass4" # provide a default

# Local variables:
Elevation = "Elevation"
Slope_Raster = "D:\Demos\FED\1406_FWS\python\intro\exerciseWork\FWSexercise.gdb\Slope_NED2017"
Steep_Slopes_Raster = "D:\Demos\FED\1406_FWS\python\intro\exerciseWork\FWSexercise.gdb\Reclass_Slop5"

# Process: Slope
arcpy.gp.Slope_sa(Elevation, Slope_Raster, "DEGREE", "1")

# Process: Reclassify
arcpy.gp.Reclassify_sa(Slope_Raster, "Value", "10 15 500;15 1000 1000", Steep_Slopes_Raster, "NODATA")

# Process: Raster to Polygon
arcpy.RasterToPolygon_conversion(Steep_Slopes_Raster, Steep_Slope_Polygons, "SIMPLIFY", "VALUE")

```

- 12) Anything that follows a # symbol is a comment- text not actually evaluated, but meant to be read. The first 2 lines run, import arcpy and arcpy.CheckOutExtension("spatial"), load in the ArcGIS tools and makes sure that we have the license to run the Spatial Analyst tools.
 13) The next part of the script looks for a name to call the output steep slope polygons when we run this at the command line; if not present, it uses a default value.
 14) The part under 'Local variables' defines the location of values (like the location of the elevation data) we need to run the script. This is a part that we almost always need to change to make a script work. Update the 'Elevation' value to look at the 'Elevation_WM' raster in the exercise GDB. Insert a line before Elevation and enter in the following:

arcpy.env.overwriteOutput = True

This will allow the script to overwrite previously created data if it has the same name as the outputs.

- 15) Save the file and close it.
 16) From the Start Menu, select 'Run' and type in cmd.exe (or type it in at the search part of the start menu). This will open up a command line window. Use the cd command to get to the workspace directory (I'll provide it in class). Type in the following:

c:\Python27\ArcGIS10.2\python.exe createSteepSlopes.py

This will run the script.

Exercise 3: Create a Script

Now that we have steep slopes and wetlands, let's combine the two to identify the slopes near the wetlands.

- 1) From the Start Menu's All Programs, select ArcGIS > Python 2.7 > IDLE. It will open with a new interactive window (much like the one we in ArcGIS Desktop).
- 2) Go to the 'File' Menu and select 'New Window'. Immediately Save the window, calling it 'identifySlopes.py' in the exercise directory. This is so IDLE will know that this is a python file.

- 3) Like the file we created by exporting a model, we should provide comments that describe what we're doing. Type in the following (using your name and today's date):

```
# identifySlopes.py
# Identifies steep slopes in the vicinity of wetlands
# Author: <Your Name>
# Date: <Today's Date>
```

- 4) Next, we'll need to load in the ArcGIS tools, so type in the following:

```
import arcpy
```

- 5) Now we need to get the parameters for this script. In this case, we need 3 items – the steep slopes, the wetlands, and the name of the feature class to write out the slopes near wetlands. To do that, we use the `arcpy.GetParameterAsText` function, which will read it from the command line. It's a good practice to also provide a meaningful default- it makes it easier to test the script as we develop it. To do that, we use the pattern shown when we exported the model- we test for a default or no value and then assign the default. Add the following lines to the script:

```
#Get Parameters
steep_slope = arcpy.GetParameterAsText(0)
if steep_slope == '#' or not steep_slope:
    steep_slope = <PATH PROVIDED IN CLASS>
```

Repeat for variables named wetlands and out_slopes.

- 6) Another good practice is to work with data in the computer's memory as much as possible. We can use a special workspace called `in_memory` to do this and copy the features into it. This also allows us to alter the data in the script without affecting the original inputs. Add the following to the script:

```
#Make copies 'in_memory' to process
arcpy.AddMessage('Copying Inputs')
steep_temp = arcpy.CopyFeatures_management(steep_slope, "in_memory\\steepSlopes")
wetlands_temp = arcpy.CopyFeatures_management(wetlands, "in_memory\\wetlands")
```

The `arcpy.AddMessage` function allows us to have the script provide some information as it runs – what's inside the function will appear either on the command line or when run in ArcGIS

- 7) We're going to be doing a series of selections against the steep slope layer. To do that, we need to create a Feature Layer (which is what you normally work with in ArcMap) from the Feature Class (the data). To do that, add the following lines:

```
#Make Feature Layer
arcpy.MakeFeatureLayer_management(steep_temp, "slopes")
```

- 8) There are a few different ways we could select the slopes; the way we'll do it here is:
 - a. Select all of the slopes of a given distance requirement (50 or 100)
 - b. Remove from the selection all the ones that meet the distance requirement
 - c. Delete the remaining (non-matching) features for a given distance

9) To accomplish this process, we need to preset the distances. To do that we'll create a list variable – that will allow us to use a for loop to do same commands with different values:

```
#Values are 50 & 100 (feet)
```

```
buffer_range = [50,100]
```

10) Now we write the instructions to handle the distance. Note that we'll be converting the distance to strings for most operations:

```
for distance in buffer_range:
```

```
    arcpy.AddMessage('Looking for slopes with a protection distance of ' +  
str(distance) + ' feet')
```

```
    arcpy.SelectLayerByAttribute_management("slopes", "NEW_SELECTION", "gridcode"  
= ' + str(distance))
```

```
    arcpy.SelectLayerByLocation_management("slopes", "INTERSECT", wetlands_temp,  
str(distance) + " Feet", "REMOVE_FROM_SELECTION")
```

```
    arcpy.DeleteFeatures_management("slopes")
```

11) After we delete the non-matching features, we need to write the output to the feature class specified.

```
arcpy.AddMessage('Writing Output')
```

```
arcpy.CopyFeatures_management(steepest_temp, out_slopes)
```

12) Save the script and attempt to run it.

For reference, it should look something like this:

```
# identifySlopes.py
# Identifies steep slopes in the vicinity of wetlands
# Author: James Tedrick
# Date: 6/16/2014

import arcpy

#Get Parameters
steep_slope = arcpy.GetParameterAsText(0)
if steep_slope == '#' or not steep_slope:
    steep_slope = "D:\\Demos\\FED\\1406_FWS\\python\\intro\\exerciseWork\\FWSexercise.gdb\\

wetlands = arcpy.GetParameterAsText(1)
if wetlands == '#' or not wetlands:
    wetlands = "D:\\Demos\\FED\\1406_FWS\\python\\intro\\exerciseWork\\FWSexercise.gdb\\Wet

out_slopes = arcpy.GetParameterAsText(2)
if out_slopes == '#' or not out_slopes:
    out_slopes = "D:\\Demos\\FED\\1406_FWS\\python\\intro\\exerciseWork\\FWSexercise.gdb\\t

#Make copies 'in_memory' to process
arcpy.AddMessage('Copying Inputs')
steep_temp = arcpy.CopyFeatures_management(steep_slope, "in_memory\\steepSlopes")
wetlands_temp = arcpy.CopyFeatures_management(wetlands, "in_memory\\wetlands")

#Make Feature Layer
arcpy.MakeFeatureLayer_management(steep_temp, "slopes")

|
#To get the features, we'll:
# 1) For each distance, select the distance
# 2) Remove from the selection the features that are in distance
# 3) Delete the remaining (selected) features

#Values are 50 & 100 (feet)
buffer_range = [50,100]

for distance in buffer_range:
    arcpy.AddMessage('Looking for slopes with a protection distance of ' + str(distance) +
    arcpy.SelectLayerByAttribute_management("slopes", "NEW_SELECTION", "gridcode" = ' + st
    arcpy.SelectLayerByLocation_management("slopes", "INTERSECT", wetlands_temp, str(distan
    arcpy.DeleteFeatures_management("slopes")

arcpy.AddMessage('Writing Output')
arcpy.CopyFeatures_management(steep_temp, out_slopes)
```

Exercise 4: Making a Script Tool

Now that we have a working script, let's add it to the Toolbox so we can run it in ArcMap

- 1) Right-click on the toolbox in the exercise directory. Select 'Add > Script'
- 2) For the Name, give 'IdentifySlope' and for the Label 'Identify Slopes near Wetlands'. In the Description, type in 'Identify Steep Slopes near wetlands based on proximity distance.' Press Next.
- 3) For the Script File, select the identifySlopes.py file you created. Click Next.
- 4) This screen is where we specify the parameters of the script. Add the following and then press Finish:

Display Name	Data Type	Direction
Steep Slope Polygons	Feature Class	Input
Wetland Polygons	Feature Class	Input
Slopes within Distance	Feature Class	Output

- 5) Run the script by double-clicking on it like other ArcToolbox Tools.
- 6) In ArcToolbox, take a look at the Data Management > Photos toolset or the Conversion Tools > Excel toolset. Note that the tools in those toolsets are also Python Tools. You can open them by right-clicking and selecting 'Edit' (You can't actually modify them, though).