Exercise 5. Building ArcGIS Tools using Python

GIS in Water Resources, Fall 2014

Prepared by Anthony Castronova

Purpose

The purpose of this exercise is to illustrate how to build ArcGIS tools using the Python programming language. This exercise will guide you through the processes of setting up Python on your computer, installing required libraries, collecting data via ArcGIS services, and building an ArcGIS tool. The purpose of the ArcGIS tool is to provide you with an example of how to manipulate shapefiles, iterate over raster datasets, execute native ArcGIS tools, as well as define ArcGIS tool parameters. Overall, it will provide guidance on how to build your own ArcGIS tool. The tool outlined in this exercise will trace a user-defined point downstream until it hits a watershed outlet.

Learning Objectives

- Understand how to setup and use Python on your own computer
- Students should be capable of basic shapefile and raster manipulation using the Python programming language.
- The ability to extend ArcGIS tools to include custom algorithms
- Understand how to develop a Python script that operates within the ArcGIS toolbox and utilizes input parameters from a user interface.

Computer and Data Requirements

To carry out this exercise, you need to have a computer that runs ArcGIS 10.2 or higher and includes the Spatial Analyst extension. No data is required to start this exercise. All the necessary data will be extracted from ArcGIS.com services. To use these services you need an ArcGIS.com account that has been linked to an ArcGIS license.

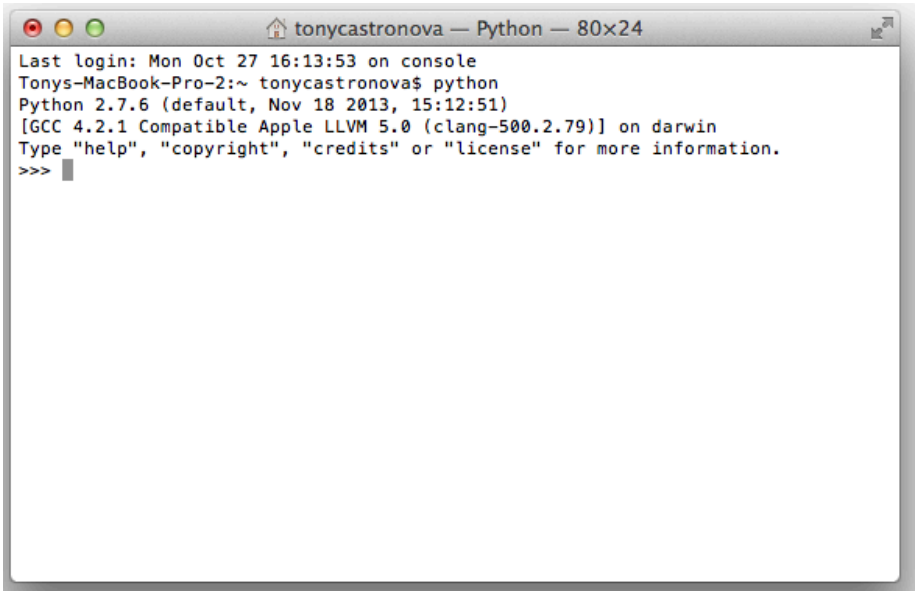
This exercise is divided into the following activities:

1. Setting up Python and installing 3rd party libraries
2. Data Collection
3. Model Building and Scripting

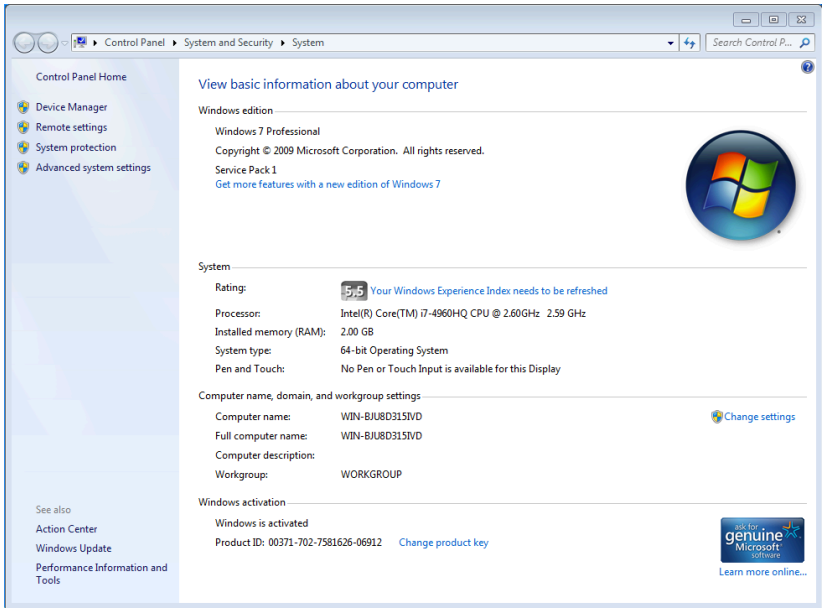
Part 1: Setting up Python and Installing 3rd party libraries

If you have ArcGIS 10.2 installed on your computer, then you already have Python 2.7 installed as well. However, you **PATH** variables may also need to be adjusted so that Python is recognized at the command-line. This is purely for running python outside of ArcGIS, but it is useful for other applications such as integrated development environments. Note: you will need administrative privileges to modify the **PATH** on you computer.

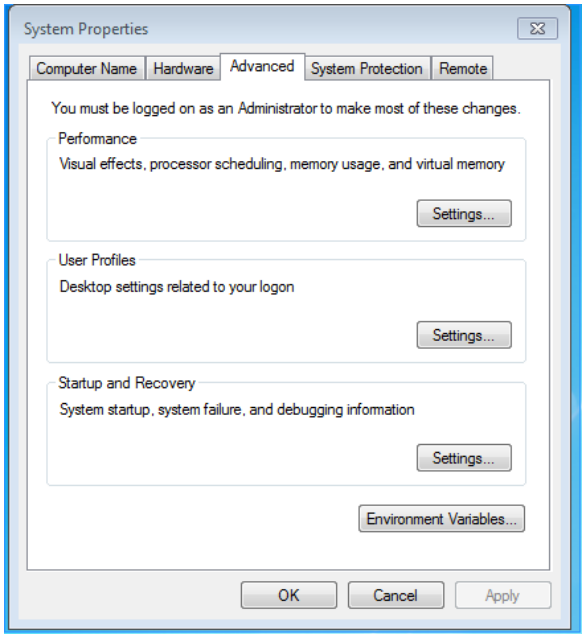
Check to see if Python is already in your **PATH** by opening the command prompt and typing **python**. If you get something like the following then you can skip down to installing 3rd party libraries.



Otherwise, click on the **start menu**, right click on **Computer,** and select **Properties**.



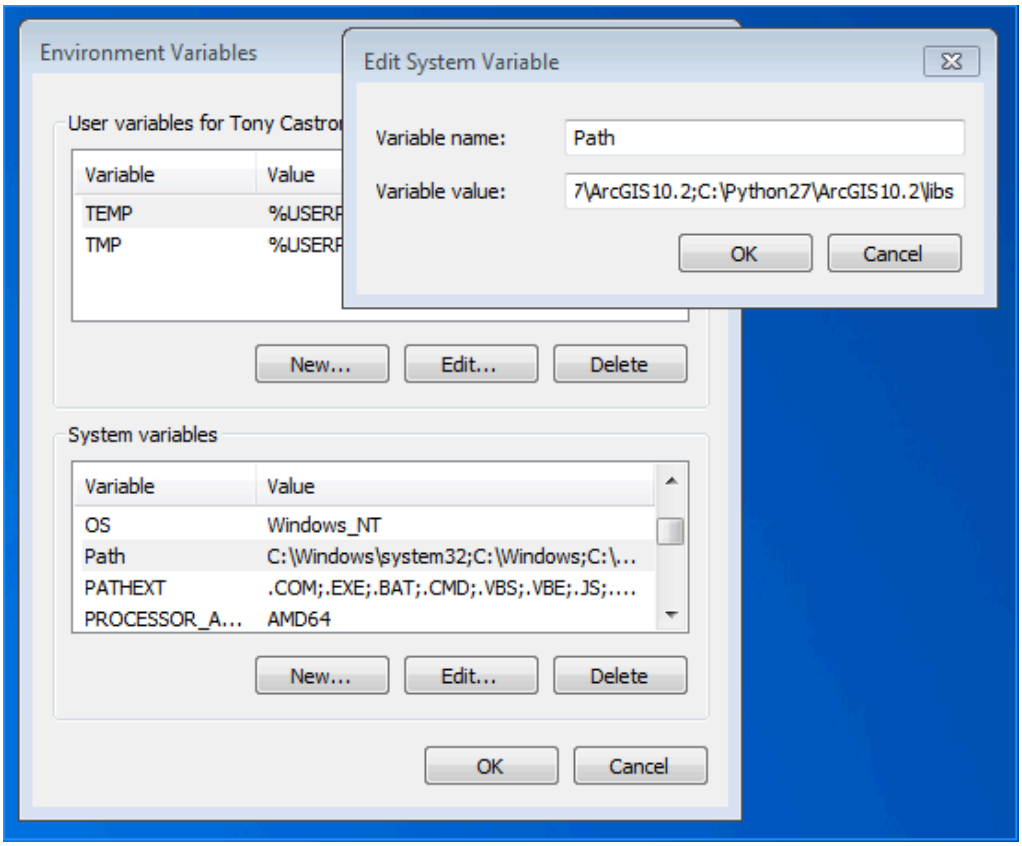
Next, select **Advance System Settings**. This will open the system properties window (below).



Click **Environmental Variables.** Find the variable labeled **Path** in the bottom window and select **edit**. Proceed with caution: Deleting items from the variable value field can effect your application and OS settings negatively.

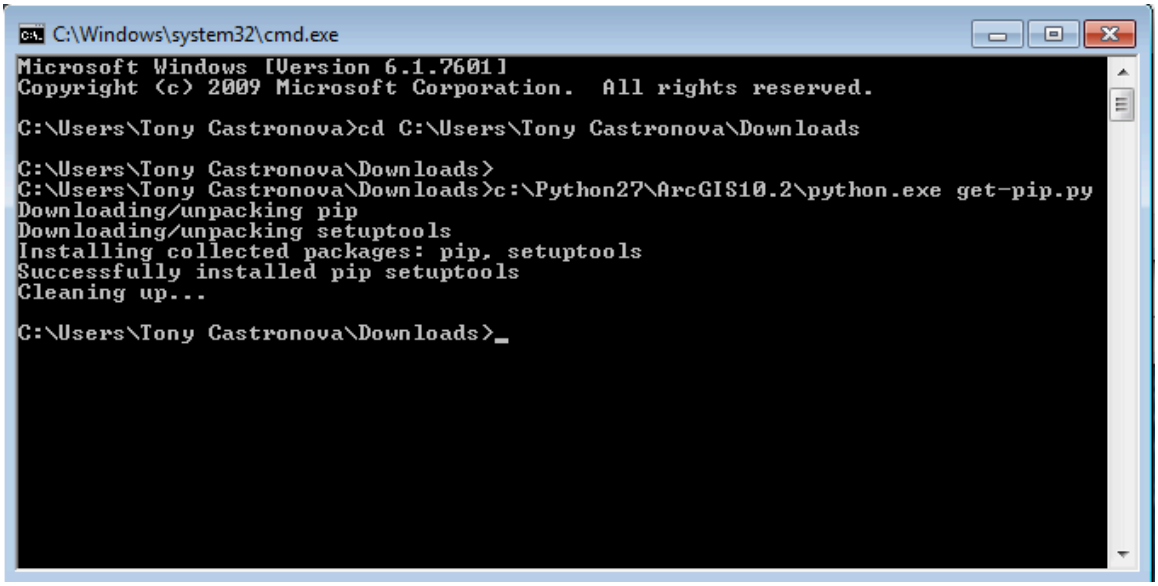
Add the following paths at the end of the variable value textbox (replace with paths on your computer!!!). Make sure you have semicolons between each path (including the one in the front)

;C:\Python27\ArcGIS10.2\Scripts;C:\Python27\ArcGIS10.2;C:\Python27\ArcGIS10.2\libs;

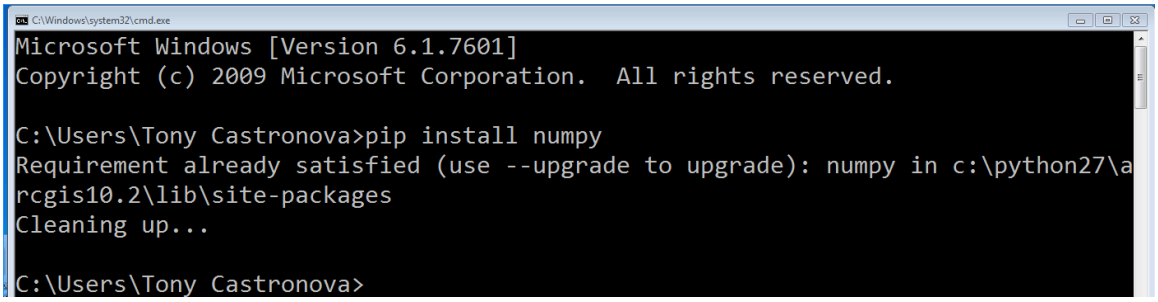


Open an **NEW** command prompt, type **python** and enter. You should now see that python has launched successfully.

Next, we can install a python package manager that will make installing 3rd party applications very easy. Navigate to https://pip.pypa.io/en/latest/installing.html and download get-pip.py. Navigate to this directory using the command line.
Type **python get-pip.py**. This will install the pip application.

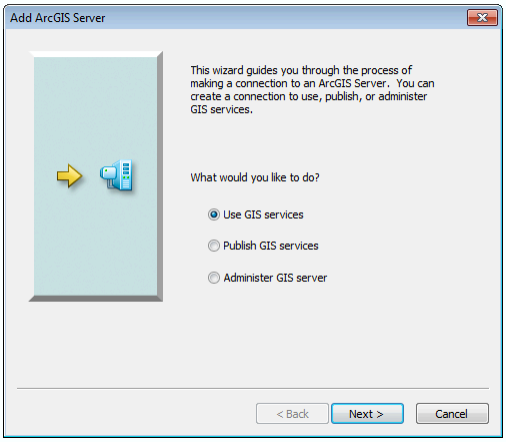


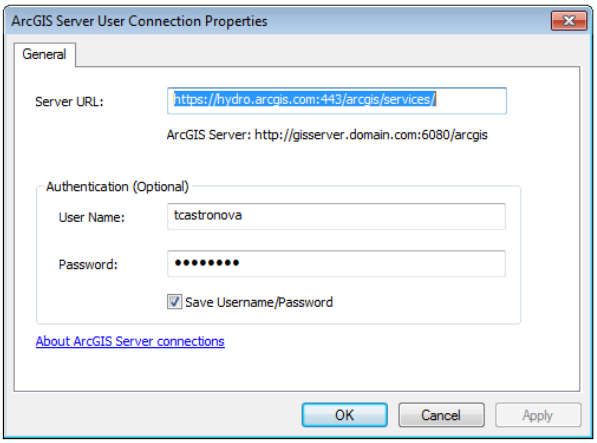
Install numpy using PIP: **pip install numpy**. Alternatively, you can download and install it using an msi (http://sourceforge.net/projects/numpy/files/NumPy/) or bundled with SciPY (http://www.scipy.org/Installing_SciPy).



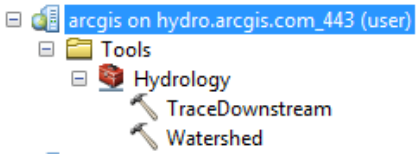
Part 2: Data Collection

Connect to the ArcGIS hydrology server. We will use this to delineate a watershed.

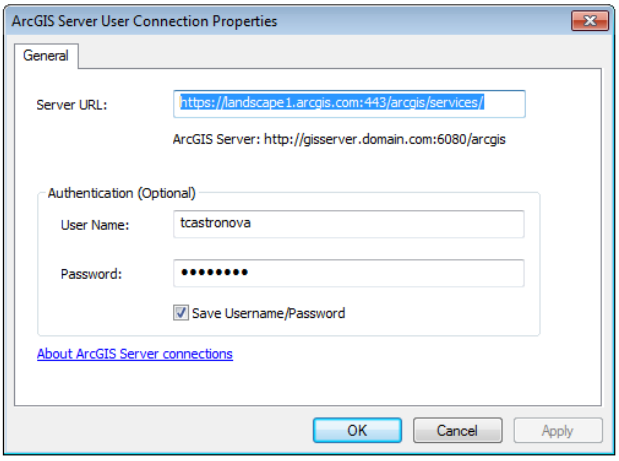


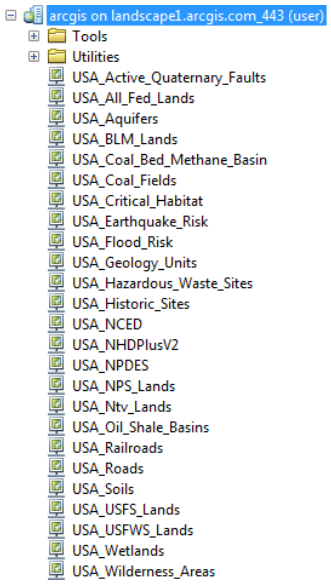


If added correctly, you should see the following tools listed in ArcCatalog.

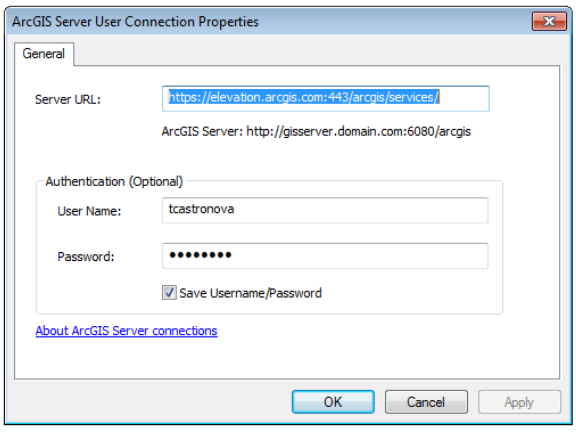


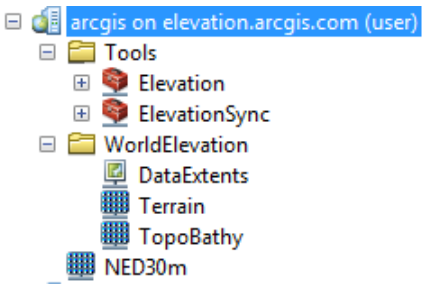
Next, add a connection to the ArcGIS landscape1 server. We will use this web service to download and visualize National Hydrography Dataset (version 2) rivers. Use https://landscape1.arcgis.com/arcgis/services as the URL. If added correctly, you will see long list of datasets under the landscape1 service in ArcCatalog.





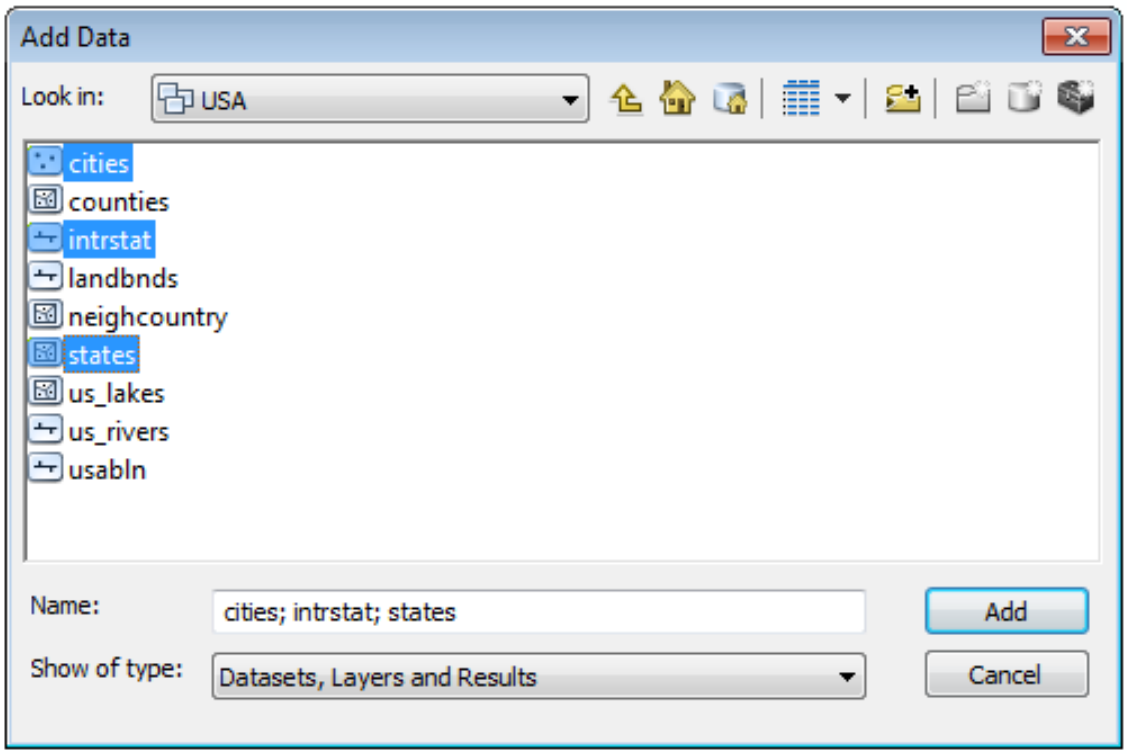
Finally, connect to the ArcGIS elevation web service. This will be used to downloading elevation data for the exercise. Use http://elevation.arcgis.com/arcgis/services as the URL. If added correctly, you will see a short list of tools and data available under the elevation service in ArcCatalog.



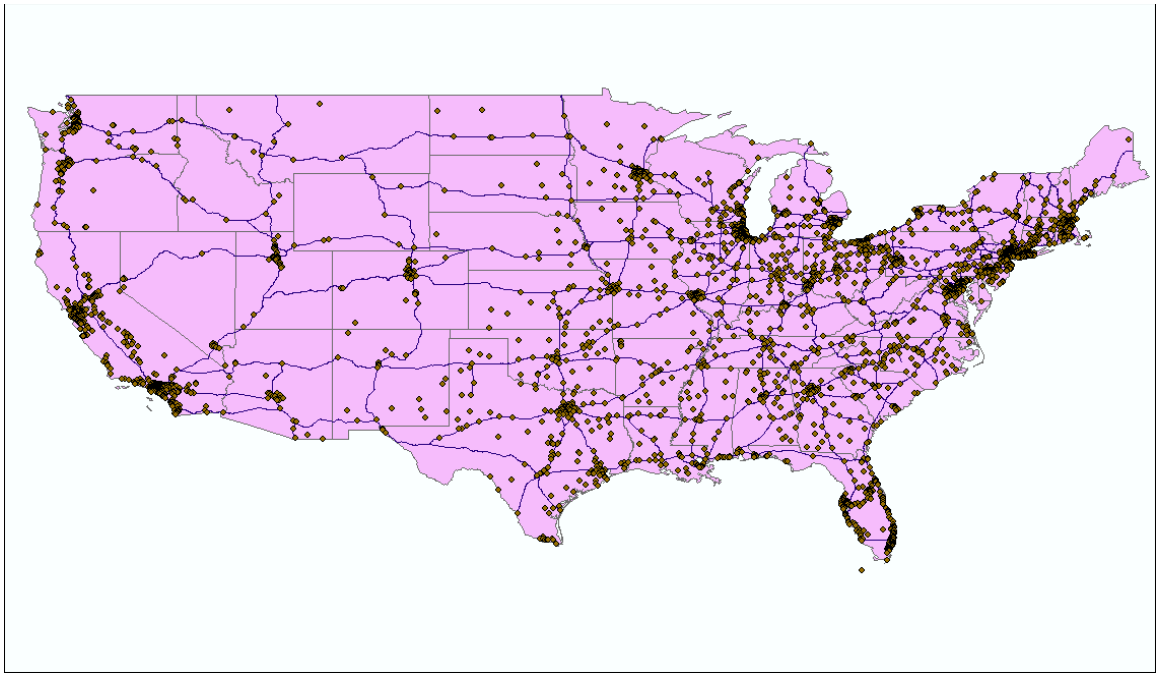


Add some template data so that we can zoom into the location that we would like to download data. Select the **Add Data** button:

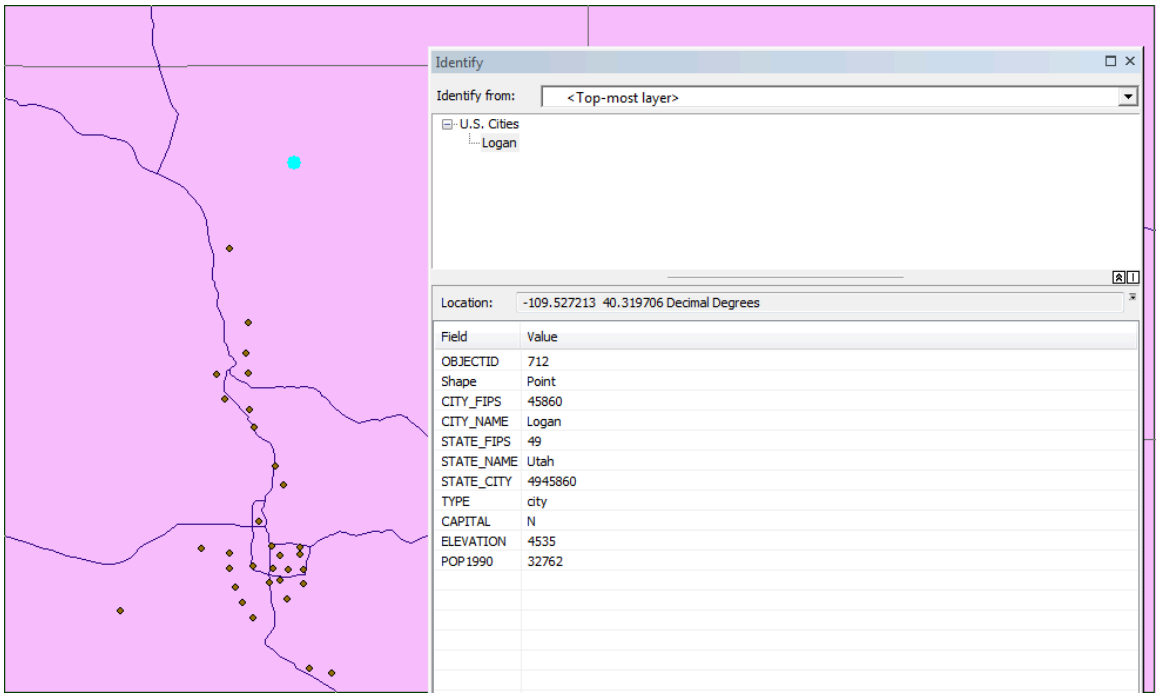
Navigate to the ArcGIS template data directory (C:\Program Files (x86)\ArcGIS\Desktop10.2\TemplateData\TemplateData.gdb\USA) and add US cities, interstates, and states.



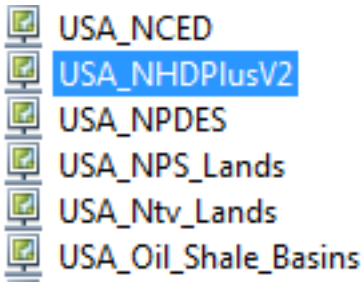
The map should now look like this:



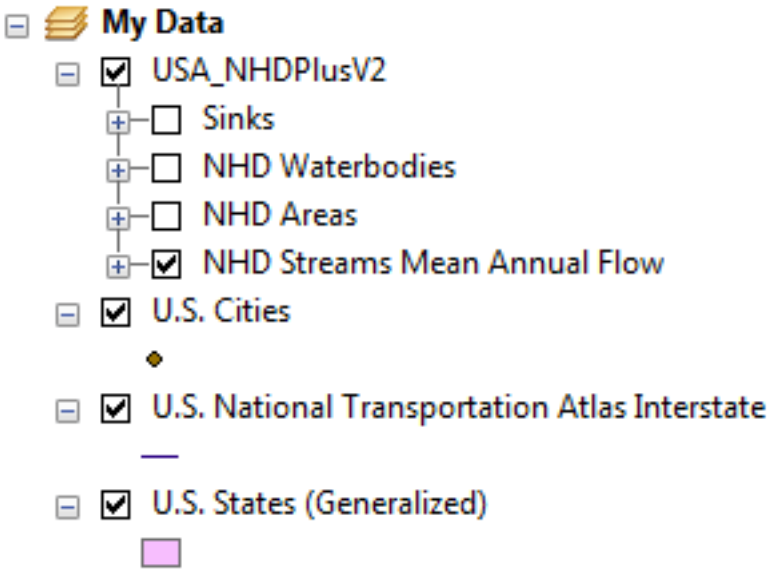
Zoom into Logan, UT. Use the **Identify** tool to determine which of these dots is Logan. This will give us an idea of where we are, before we start loading ArcGIS web service datasets.



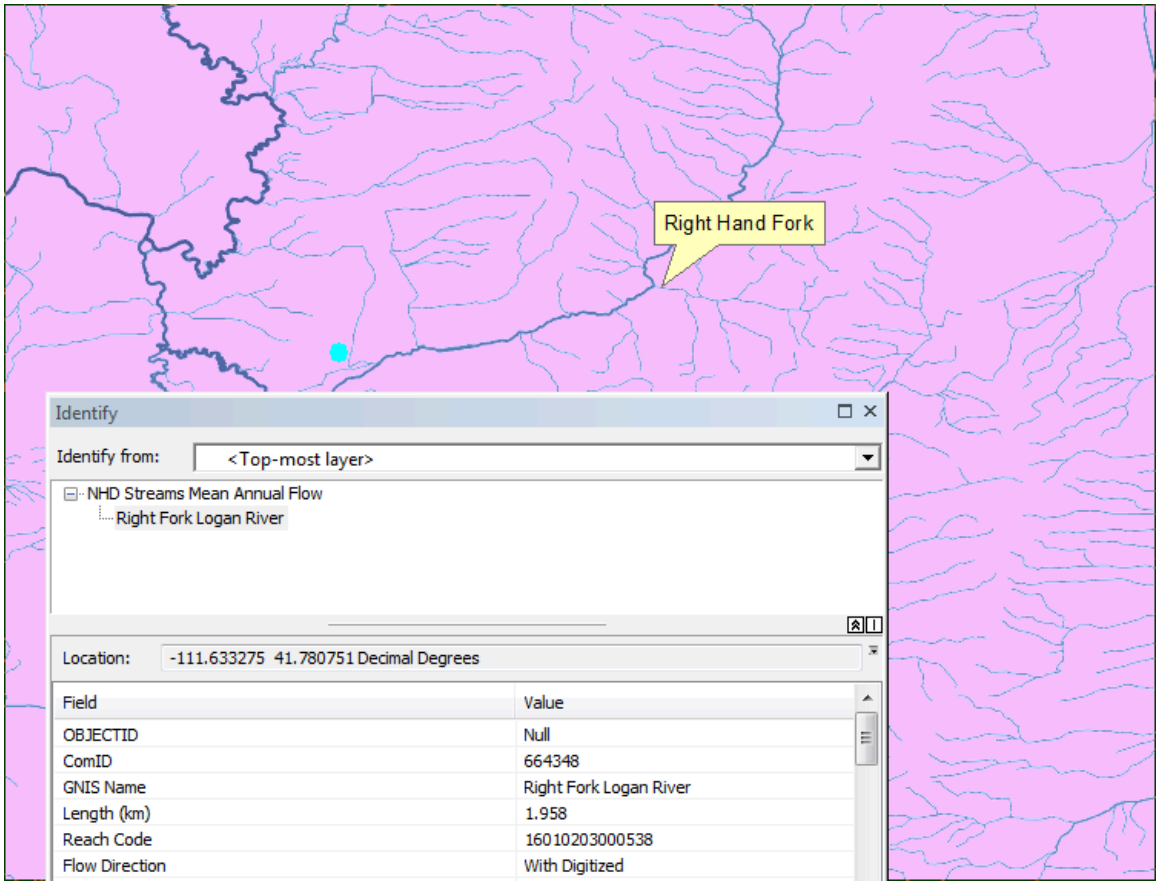
Add the NHDPlus (version 2) data set from the landscape1.arcgis.com web service.



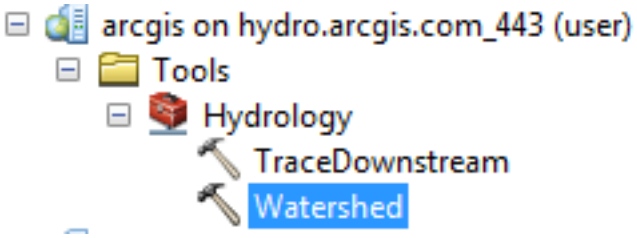
We are only interested in the stream data, so turn off all NHD layers except Streams. This will help speed up the data load time. The layers in your table of contents should look like this:



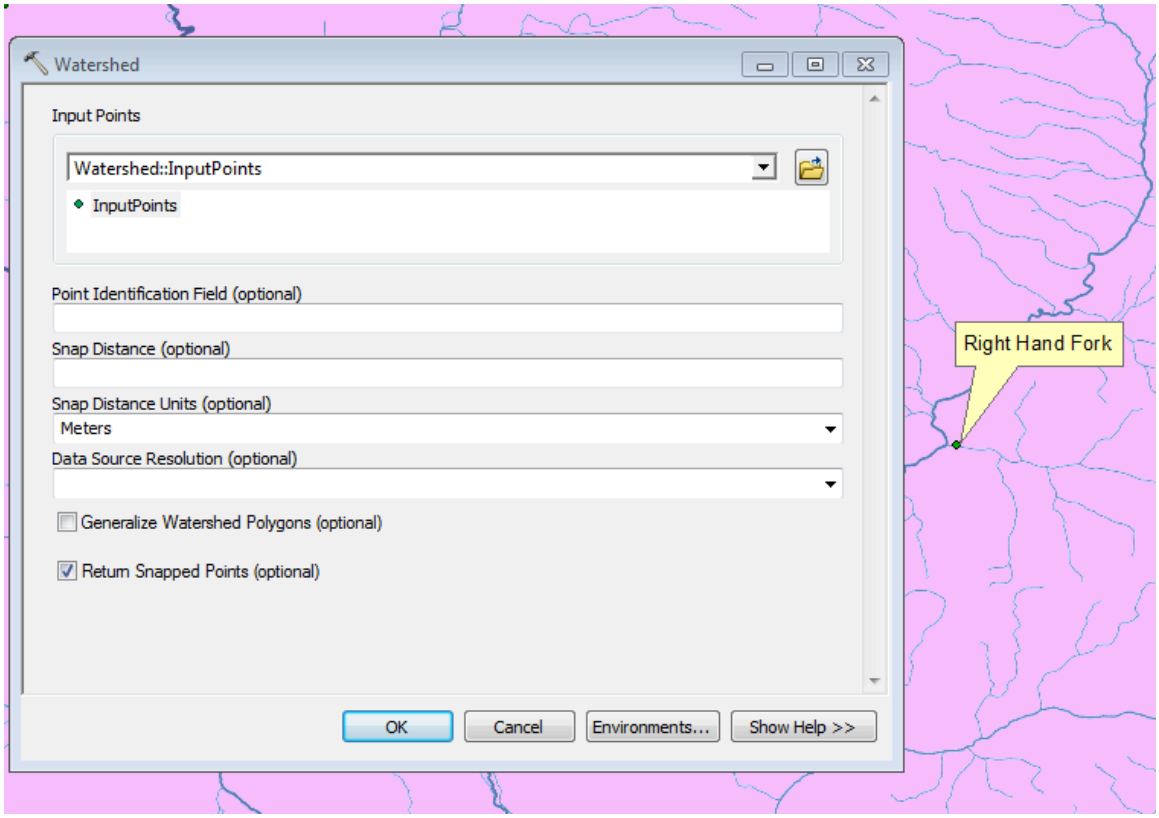
Now that we have the NHD rivers loaded, we can zoom into Right Hand Fork.



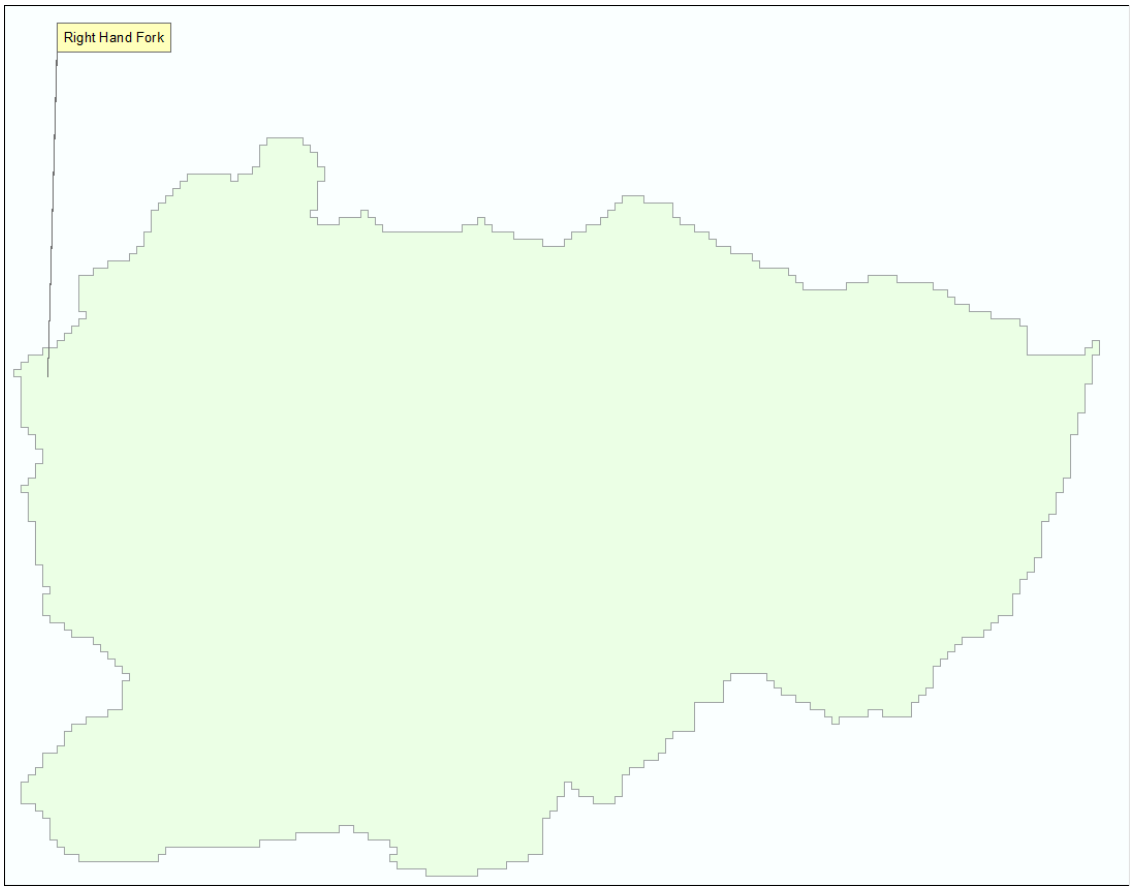
To delineate a watershed at Right Hand Fork, we will use the ArcGIS online watershed delineation tool. Double click on the ArcGIS server watershed tool.



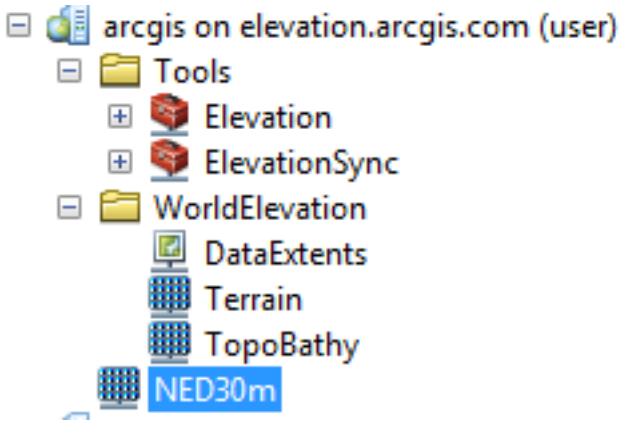
Select an input point near the outlet of Right Hand Fork (see green dot on map). Don't get too close to the Logan river (downstream), or the delineation tool will snap the outlet to the wrong reach. To ensure that this does not happen, you may have to adjust the snap distance (try 100 meters)



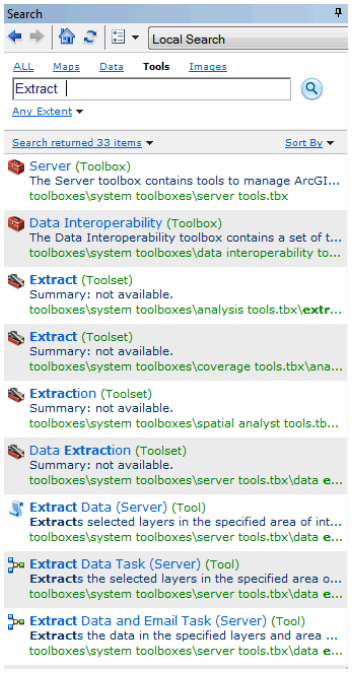
This operation will result in the Right Hand Fork watershed. Go ahead and turn off all unnecessary layers and change the watershed color to something more meaningful.

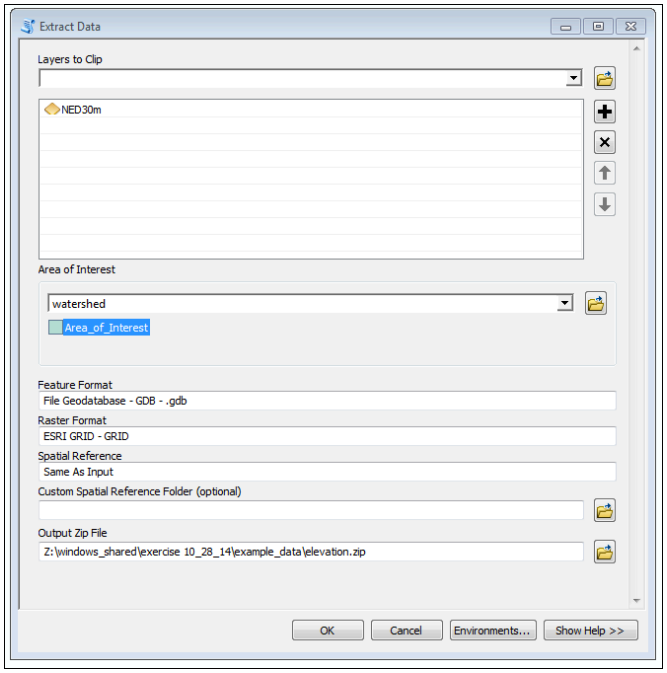


Add NED30m elevation from the elevation.arcgis.com server.



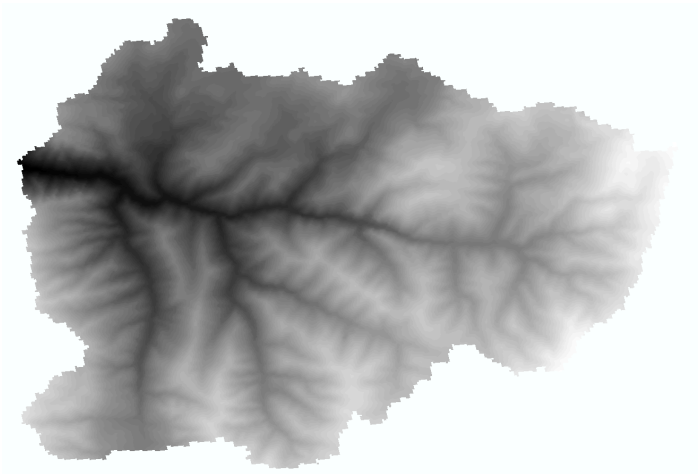
Next we want to extract the elevation data within the boundary of our watershed. This will make future data processing faster since we will be using a small subset of the national elevation dataset. In addition, this file will be stored locally so we won't need an Internet connection to perform our processing tasks. To do this, open the search menu and enter "Extract". Make sure to choose the search by "Tools" option above the search textbox. This will limit the search results ArcGIS tools. Since we are dealing with elevation data from an ArcGIS server, we want to select the "Extract Data (server)" tool.





Select the NED 30m elevation raster as the layer to clip. The Area of Interest that will be used to extract the data (i.e. cookie cutter) should be the watershed that you delineated in previous steps. Leave the default options for Feature Format, Raster Format, Spatial Reference, and Custom Spatial Reference Folder. Specify an output ZIP file where the extracted data will be saved.

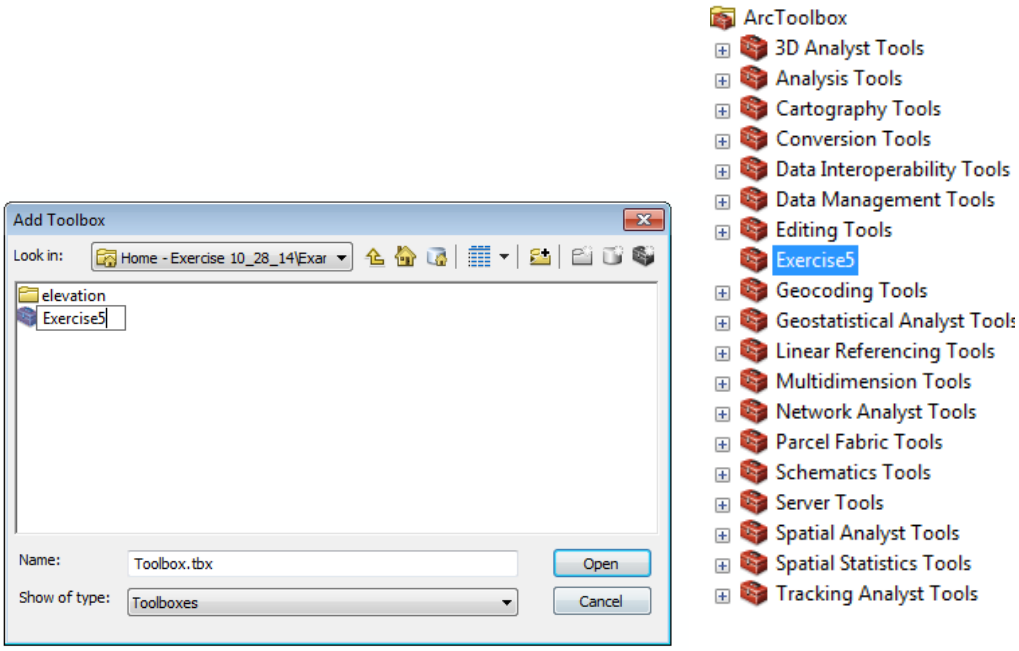
Open Windows Explorer and navigate to the directory of your output ZIP. Extract the contents, and you should now have an elevation dataset that covers only the watershed area.



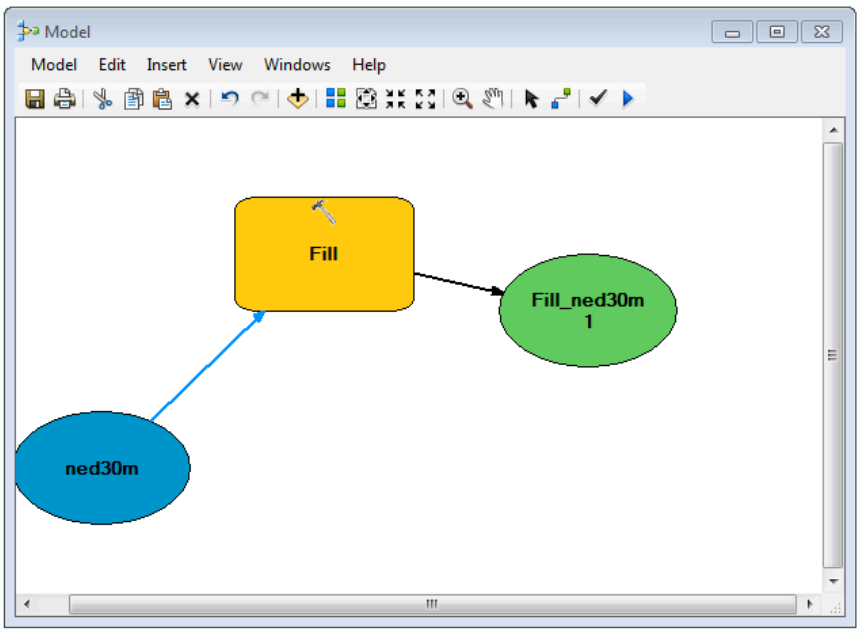
Part 3: Model Building and Scripting

The goal of our scripting tool is to trace any point within the watershed downstream to the watershed outlet. This can later be modified to provide statistics regarding the flow path. This example will demonstrate (1) how ArcGIS tools can be used to create a custom model, (2) how to include custom data processing and functionality, and (3) how to build the ArcGIS tool interface for a custom tool.

Activate the ArcToolbox by clicking  . Create a new toolbox by right clicking inside the window and selecting Add Toolbox from the context menu. This will open a dialog for you to search for an existing toolbox. Instead, navigate to any directory that you like and select the create New Toolbox button in the top right corner  .



After creating your toolbox (i.e. Exercise 5), right click on it and select New -> Model. You will end up with an empty model. Drag and drop the Fill tool onto the canvas, along with the clipped elevation raster.



From the menu, select Model -> Export -> To Python Script. Open the export Python file to view the code that was written for us by ArcGIS. This is an easy way to extend a model that you have already created.

```
# -*- coding: utf-8 -*-
# ---------------------------------------------------------------------------
# trace_point_downstream.py
# Created on: 2014-10-27 14:23:28.00000
#   (generated by ArcGIS/ModelBuilder)
# Description:
# ---------------------------------------------------------------------------

# Import arcpy module
import arcpy

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


# Local variables:
ned30m = "ned30m"
Fill_ned30m1 = "C:\\Users\\Tony Castronova\\Documents\\ArcGIS\\Default.gdb\\Fill_ned30m1"

# Process: Fill
arcpy.gp.Fill_sa(ned30m, Fill_ned30m1, "")
```

Notice that there are some strange variable names. Lets modify this code so that the variable names make a little more sense and fix the file paths. Also import the **numpy**, and **math** libraries which we will need later.

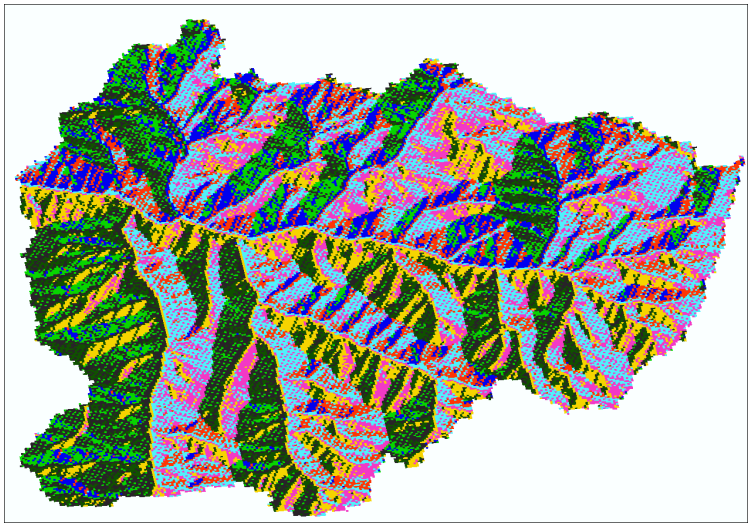
```
# Import arcpy module
print 'importing arcpy (this takes a while)...'
import arcpy
from arcpy import env
from arcpy.sa import *
import numpy
import math

# Check out any necessary licenses
arcpy.CheckOutExtension("spatial")
env.overwriteOutput = True

# Local variables:
ned30m = "Z:/windows_shared/exercise 10_28_14/example_data/elevation/zipfolder/ned30m"
fill_outpath = "Z:/windows_shared/exercise 10_28_14/example_data/fill"

# Process: Fill
print 'Run Fill!'
outFill = Fill(ned30m, "")
outFill.save(fill_outpath)
#arcpy.gp.Fill_sa(ned30m, fill_outpath, "")

print 'done'
```

Lets run this code and see what kind of output we get. If the script ran successfully, we should have a new raster called **fill** that can be opened in ArcMap. Note: the original script used the **gp.Fill_sa** tool whereas the documentation states that we should use the **arcpy.sa.Fill** tool. If you encounter this, I suggest that you use the tools outlined in the ArcGIS documentation.

Next, lets calculate flow direction. To determine the syntax for this operation we can google "ArcGIS Flow Direction":
http://help.arcgis.com/en/arcgisdesktop/10.0/help/index.html#//009z00000052000000.htm

```
# Import arcpy module
print 'importing arcpy (this takes a while)...'
import arcpy
from arcpy import env
from arcpy.sa import *
import numpy
import math

# Check out any necessary licenses
arcpy.CheckOutExtension("spatial")
env.overwriteOutput = True

# Local variables:
ned30m = "Z:/windows_shared/exercise 10_28_14/example_data/elevation/zipfolder/ned30m"
fill_outpath = "Z:/windows_shared/exercise 10_28_14/example_data/fill"
fdr_outpath = "Z:/windows_shared/exercise 10_28_14/example_data/fdr"    # Added

# Process: Fill
print 'Run Fill!'
outFill = Fill(ned30m, "")
#outFill.save(fill_outpath)                                    # Modified

# Prcess: FDR
print 'Run FDR!'                                               # Added
outFlowDirection = FlowDirection(outFill, "NORMAL")            # Added
outFlowDirection.save(fdr_outpath)                             # Added


print 'done'
```

Notice that the output from the fill operation was not **Saved**, but it can still be used in the following step! This is because it is saved temporarily in memory. We can utilize this feature to "hide" intermediary processing outputs. Lets look at the output from the flow direction process.



Now that we have some of the basic raster processing done, lets create a point that can be traced to the outlet. This will be hardcoded for now, but we can change it to a user input later.

```
# Import arcpy module
print 'importing arcpy (this takes a while)...'
import arcpy
from arcpy import env
from arcpy.sa import *
import numpy
import math

# Check out any necessary licenses
arcpy.CheckOutExtension("spatial")
env.overwriteOutput = True

# Local variables:
ned30m = "Z:/windows_shared/exercise 10_28_14/example_data/elevation/zipfolder/ned30m"
fill_outpath = "Z:/windows_shared/exercise 10_28_14/example_data/fill"
fdr_outpath = "Z:/windows_shared/exercise 10_28_14/example_data/fdr"

# create a point object                                                    # Added
my_x = -1216071.141                                                        # Added
my_y = 307660.098                                                          # Added
pnt = arcpy.Point(my_x, my_y)                                              # Added

# Process: Fill
print 'Run Fill!'
outFill = Fill(ned30m, "")
#outFill.save(fill_outpath)

# Prcess: FDR
print 'Run FDR!'
outFlowDirection = FlowDirection(outFill, "NORMAL")
outFlowDirection.save(fdr_outpath)
```

In order to relate this point coordinate with the raster data, we need to do two things: (1) represent the raster grids as arrays of data, and (2) convert the x,y point coordinate into array indices. To convert the raster grids (i.e. **fill** and **fdr**) into arrays, we use the numpy library, specifically RastertoNumPyArray.

```
# Import arcpy module
print 'importing arcpy (this takes a while)...'
import arcpy
from arcpy import env
from arcpy.sa import *
import numpy
import math

# Check out any necessary licenses
arcpy.CheckOutExtension("spatial")
env.overwriteOutput = True

# Local variables:
ned30m = "Z:/windows_shared/exercise 10_28_14/example_data/elevation/zipfolder/ned30m"
fill_outpath = "Z:/windows_shared/exercise 10_28_14/example_data/fill"
fdr_outpath = "Z:/windows_shared/exercise 10_28_14/example_data/fdr"

# create a point object
my_x = -1216071.141
my_y = 307660.098
pnt = arcpy.Point(my_x, my_y)

# Process: Fill
print 'Run Fill!'
outFill = Fill(ned30m, "")
#outFill.save(fill_outpath)

# Prcess: FDR
print 'Run FDR!'
outFlowDirection = FlowDirection(outFill, "NORMAL")
outFlowDirection.save(fdr_outpath)

# convert rasters to arrays                                                    # Added
fdr  = arcpy.RasterToNumPyArray(outFlowDirection, nodata_to_value=0)          # Added
fill = arcpy.RasterToNumPyArray(outFill, nodata_to_value=0)                   # Added

print 'done'
```

If you print the value of the **fdr** value you will see this:

```
>>> fdr
array([[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]], dtype=uint8)
```

It looks like there are lots of 0's, however this is just because we are seeing a small subset of the data. In fact most of the cells near the edge of the raster will be zero. Lets look at some values elsewhere:

```
>>> fdr[100:110, 100:110]
array([[ 2,  4,  8,  4,  4,  4,  4,  4,  8, 16],
       [ 1,  4, 16,  4,  4,  4,  4,  4,  8,  4],
       [ 1,  2,  4,  4,  4,  4,  4,  8,  8,  4],
       [ 2,  1,  2,  2,  4,  4,  4,  8,  4,  8],
       [ 2,  1,  2,  2,  2,  4,  4,  8,  8,  8],
       [ 1,  2,  2,  1,  2,  4,  8, 16,  8, 16],
       [ 1,  2,  1,  2,  2,  4,  8, 16, 16, 32],
       [ 2,  2,  4,  2,  4,  8, 16, 16, 16, 32],
       [ 2,  1,  2,  2,  4,  8, 16, 16, 32, 16],
       [ 1,  1,  2,  1,  4, 16, 16, 32, 32, 16]], dtype=uint8)
```

Before we do anymore processing of the raster data, we need to extract some metadata that will enable us to loop over the raster cells. The numpy arrays only contain raster values, so we will need to use the ArcGIS Raster type to retrieve this information.

```
...
...

# convert rasters to arrays
fdr  = arcpy.RasterToNumPyArray(outFlowDirection, nodata_to_value=0)
fill = arcpy.RasterToNumPyArray(outFill, nodata_to_value=0)

# create raster object to get metadata
upperLeft = outFill.extent.upperLeft
ux = upperLeft.X
uy = upperLeft.Y
cell_width = outFill.meanCellWidth
cell_height = outFill.meanCellHeight

...
...
```

We can transform our point coordinates into array indices, now that we have the upper left (x,y), cell width, and cell height. This will enable us to access the raster value of the cell associated with our point.

```
...
...

# convert rasters to arrays
fdr  = arcpy.RasterToNumPyArray(outFlowDirection, nodata_to_value=0)
fill = arcpy.RasterToNumPyArray(outFill, nodata_to_value=0)

# create raster object to get metadata
upperLeft = outFill.extent.upperLeft
ux = upperLeft.X
uy = upperLeft.Y
cell_width = outFill.meanCellWidth
cell_height = outFill.meanCellHeight

# convert point coordinates into raster indices
c = abs(int((ux - pnt.X) / cell_width))
r = abs(int((uy - pnt.Y) / cell_height))
...
...
```

Lets see where our point lives in the raster array:

```
>>> (pnt.X,pnt.Y), '--->',(c,r)
((-1216071.141, 307660.098), '--->', (62, 210))
```

Now we are ready to start moving our point around within the raster. Specifically, we want to move our point from its current location (62,210) to the next downstream cell. In order to accomplish this, we need to add a function at the **top** of our script to check the value of our flow direction grid and move the point accordingly. Place this function right below the import statements.

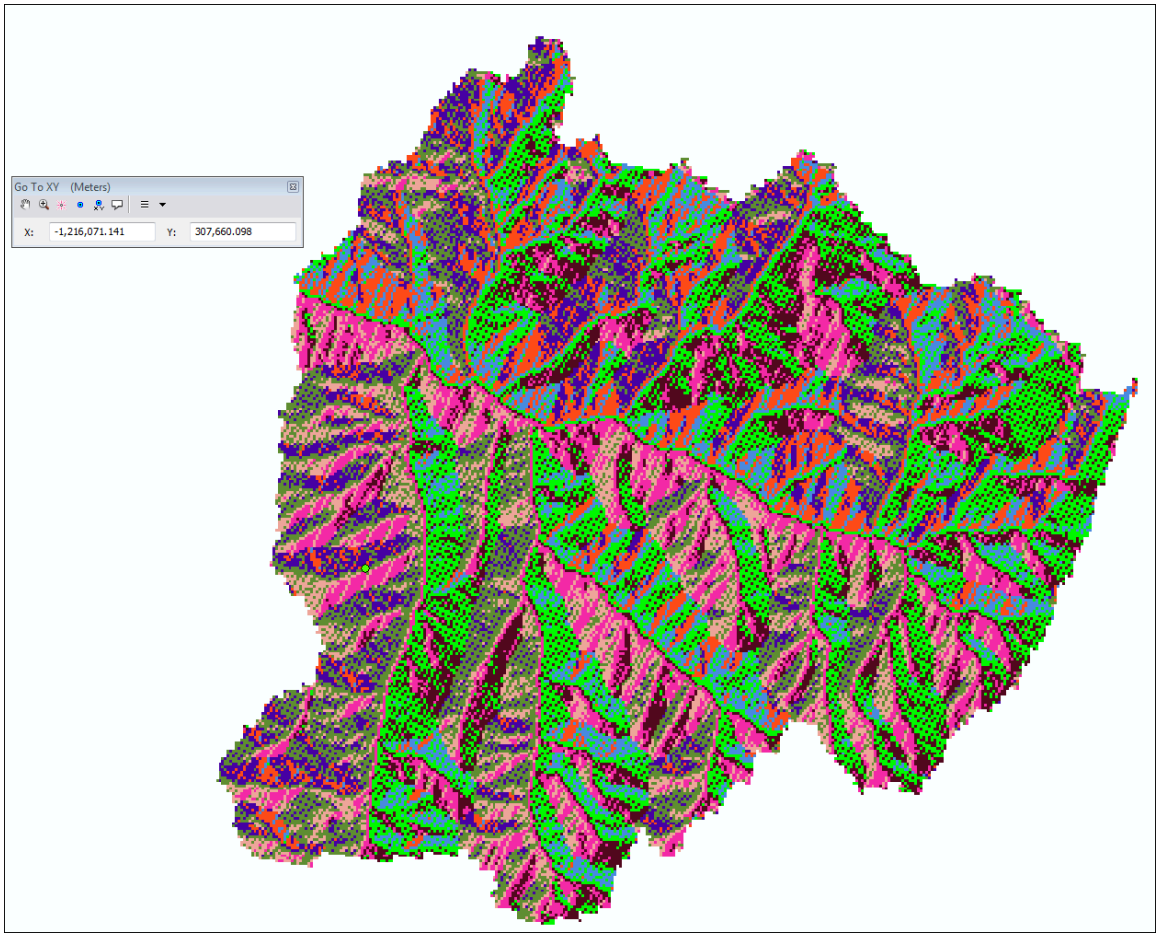
```
def move_to_next_pixel(fdr, row, col):

    # get the fdr pixel value (x,y)
    value = fdr[row, col]

    #
    #| 32 |  64  | 128 |
    #| 16 |  X   |  1  |
    #|  8 |  4   |  2  |
    #

    # move the pixel
    if value == 1:
        col += 1
    elif value == 2:
        col += 1
        row += 1
    elif value == 4:
        row += 1
    elif value == 8:
        row += 1
        col -= 1
    elif value == 16:
        col -= 1
    elif value == 32:
        row -= 1
        col -= 1
    elif value == 64:
        row -= 1
    else: #value == 128:
        row -= 1
        col += 1

    return (row, col)
```

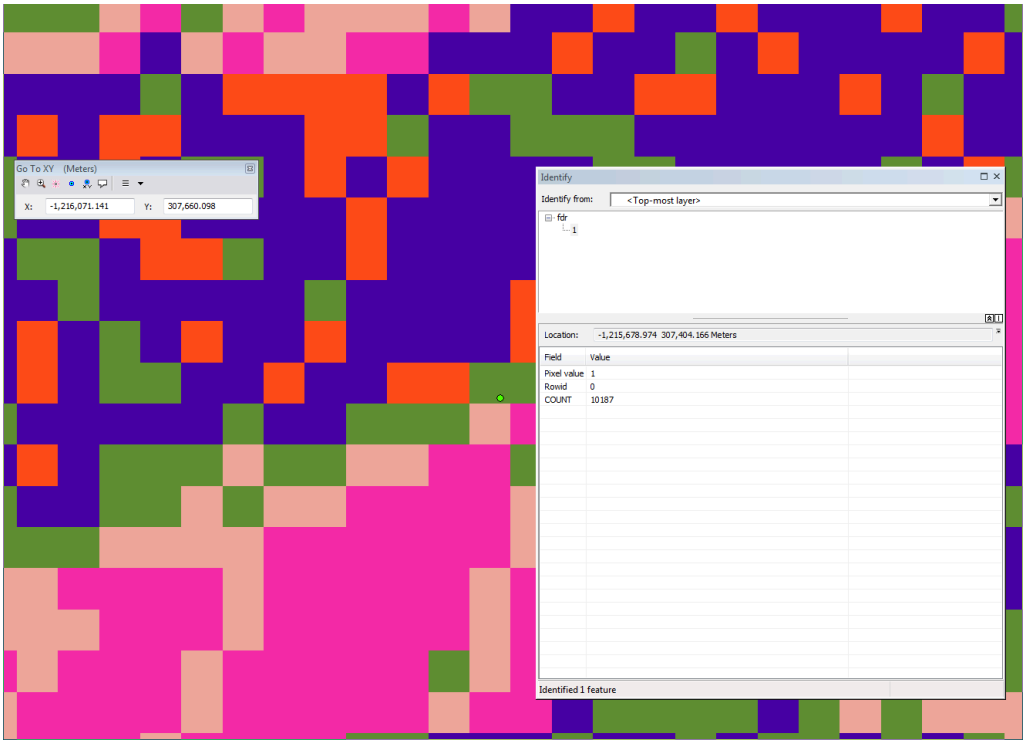
This function takes in three arguments: fdr (flow direction array), row (current row index), col (current col index). The first thing that it does is extract the value of the flow direction grid at the current (row, col) location. It then checks this value against all the possible flow direction combinations to determine the next downstream neighbor. It increments the current (row, col) pair and returns the result.

Lets pass in the coordinates of our point and see which direction our cell will flow.

```
>>> r,c
(210, 62)
>>> move_to_next_pixel(fdr, r, c)
(210, 63)
```

We can verify this by loading the flow direction raster into ArcMap.




Go To XY (Meters)
X: -1,216,071.141
Y: 307,660.098
Identify
Identify from:
<Top-most layer>
fdr
1
Location: -1,215,678.974 307,404.166 Meters
Field
Value
Pixel value 1
Rowid 0
COUNT 10187
Identified 1 feature

Lets can modify our code to repeat this process until the point moves beyond the extent of our raster grid (e.g. through the outlet). In order do so, we need to create a loop that will run until the value at location (r,c) is equal to NoDATA (in this case 0).

```
...
...

# convert rasters to arrays
fdr  = arcpy.RasterToNumPyArray(outFlowDirection, nodata_to_value=0)
fill = arcpy.RasterToNumPyArray(outFill, nodata_to_value=0)

# create raster object to get metadata
upperLeft = outFill.extent.upperLeft
ux = upperLeft.X
uy = upperLeft.Y
cell_width = outFill.meanCellWidth
cell_height = outFill.meanCellHeight

# convert point coordinates into raster indices
c = abs(int((ux - pnt.X) / cell_width))
r = abs(int((uy - pnt.Y) / cell_height))

z = fill[r,c]
while (z != 0):
    pass
...
...
```

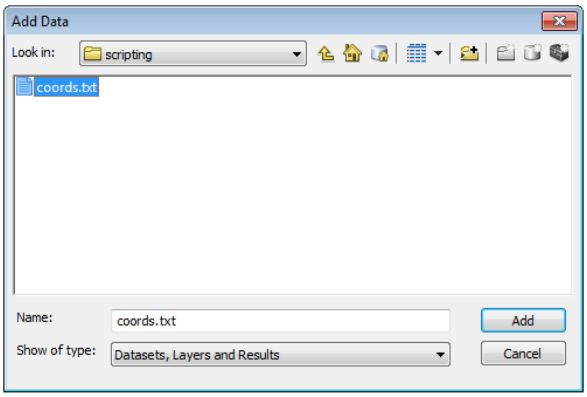
This loop will continue to run while the value of z does not equal 0 (i.e. no data value). Currently, this loop will run indefinitely because **z** is not changing inside the loop. Lets add some code to fix this by moving (r,c) to its downstream neighbor.

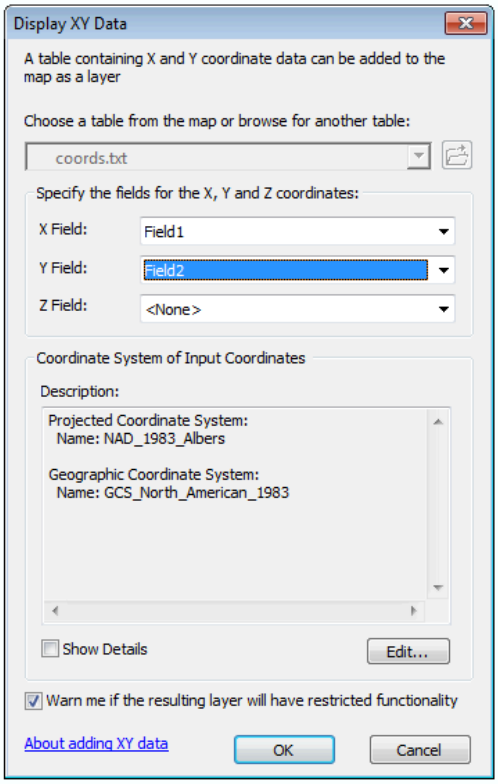
```
...
...

# convert rasters to arrays
fdr  = arcpy.RasterToNumPyArray(outFlowDirection, nodata_to_value=0)
fill = arcpy.RasterToNumPyArray(outFill, nodata_to_value=0)

# create raster object to get metadata
upperLeft = outFill.extent.upperLeft
ux = upperLeft.X
uy = upperLeft.Y
cell_width = outFill.meanCellWidth
cell_height = outFill.meanCellHeight

# convert point coordinates into raster indices
c = abs(int((ux - pnt.X) / cell_width))
r = abs(int((uy - pnt.Y) / cell_height))

z = fill[r,c]
while (z != 0):
    r,c = move_to_next_pixel(fdr, r, c)
...
...
```

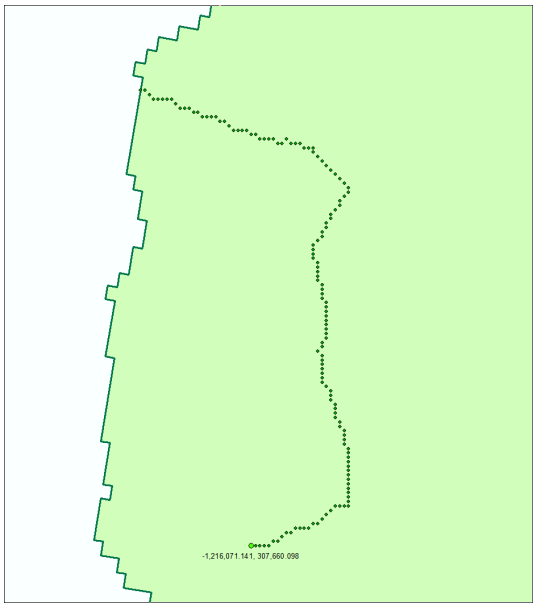
This code will move the point (r,c) to its downstream neighbor, and continue to do so until we reach the watershed outlet. Unfortunately, we have no output to visualize. Lets save these points in a list and then create a shapefile that we can visualize in ArcMap.

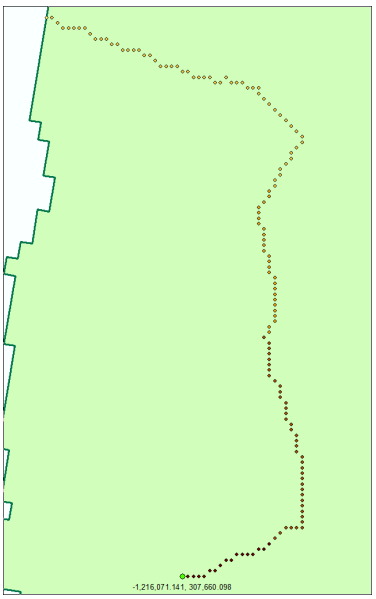
```
...
...

# convert rasters to arrays
fdr  = arcpy.RasterToNumPyArray(outFlowDirection, nodata_to_value=0)
fill = arcpy.RasterToNumPyArray(outFill, nodata_to_value=0)

# create raster object to get metadata
upperLeft = outFill.extent.upperLeft
ux = upperLeft.X
uy = upperLeft.Y
cell_width = outFill.meanCellWidth
cell_height = outFill.meanCellHeight

# convert point coordinates into raster indices
c = abs(int((ux - pnt.X) / cell_width))
r = abs(int((uy - pnt.Y) / cell_height))

z = fill[r,c]
while (z != 0):
    # move downstream
    last_r = r
    last_c = c
    r,c = move_to_next_pixel(fdr, r, c)

    # adjust x and y
    pntX += (last_c-c)*cell_width
    pntY += (last_r-r)*cell_height
    z = fill[r,c]

    # save this coordinate
    coords.append((pntX,pntY,z))

# write the output to text file
with open('coords.txt','w') as f:
    for c in coords:
        f.write('%5.5f, %5.5f, %5.5f\n' % (c[0],c[1],c[2]))
```

To visualize our output in ArcMap, add the **coords.txt** file to an ArcMap document. Right click on it and select **Display X,Y data**. Choose **Field1** as the X field and **Field 2** as the Y field. You can also symbolize these points by their elevation, **Field 3**



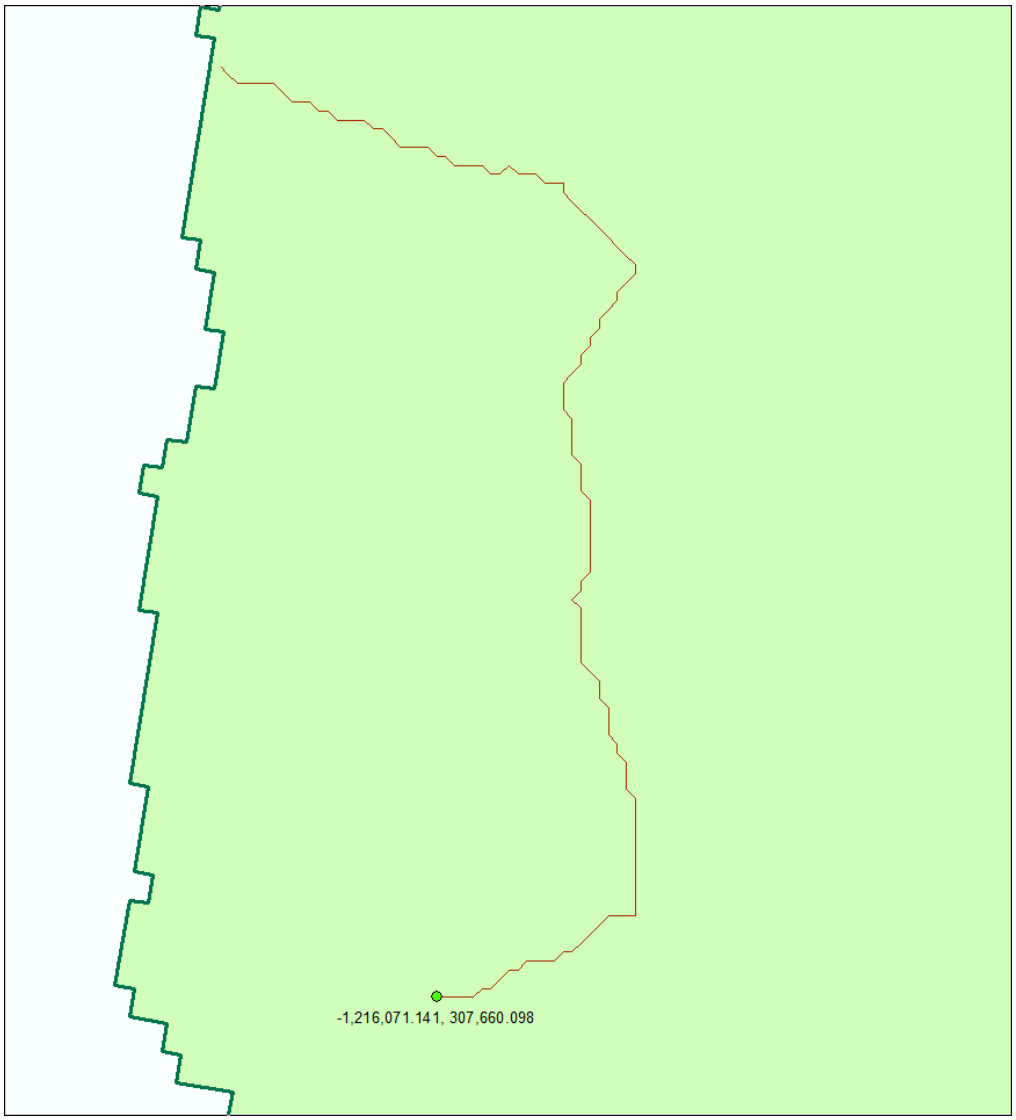






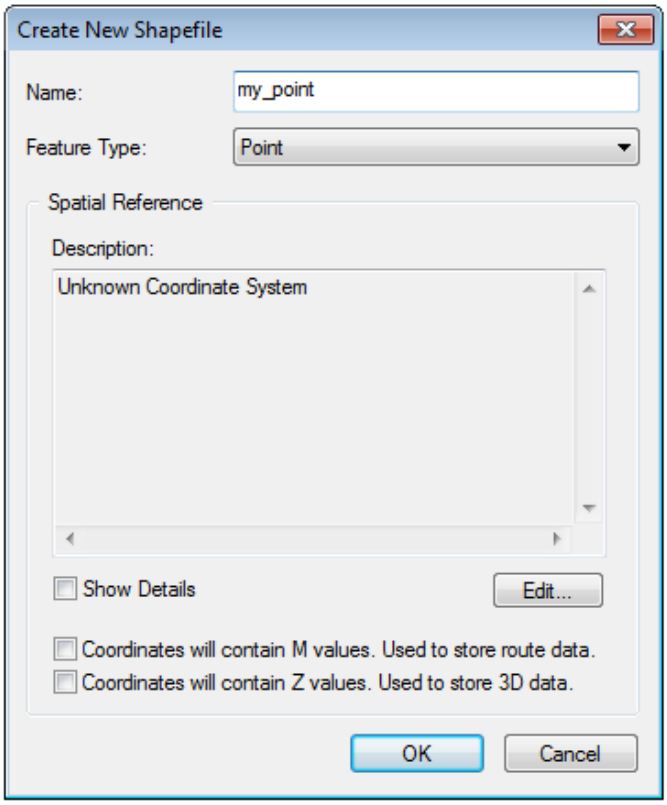
Since point text file is not an ideal output, lets format it as a PolyLine Shapefile, http://help.arcgis.com/en/arcgisdesktop/10.0/help/index.html#//001700000002p000000. In the code snippet below, we first create the polyline feature class that will hold our results. Next we loop over our coordinates are create line segments between each pair. These line segments are then added to the feature class as a polyline.

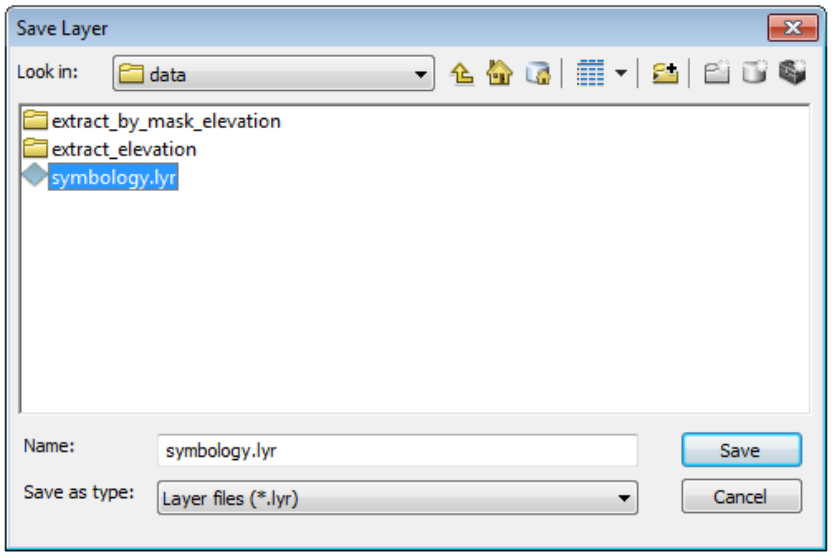
```
# create the output feature class
arcpy.CreateFeatureclass_management('.','path.shp', "POLYLINE")

# define the point and line segment objects
point = arcpy.Point()
line_seg = arcpy.Array()

featureList = []
cursor = arcpy.InsertCursor('path.shp')
feat = cursor.newRow()

for i in range(1, len(coords)-1):
    # Set X and Y for start and end points
    point.X = coords[i-1][0]
    point.Y = coords[i-1][1]
    line_seg.add(point)
    point.X = coords[i][0]
    point.Y = coords[i][1]
    line_seg.add(point)

    # Create a Polyline object based on the array of points
    polyline = arcpy.Polyline(line_seg)

    # Clear the array for future use
    line_seg.removeAll()

    # Append to the list of Polyline objects
    featureList.append(polyline)

    # Insert the feature
    feat.shape = polyline
    cursor.insertRow(feat)
del feat
del cursor
```


-1,216,071.141, 307,660.098

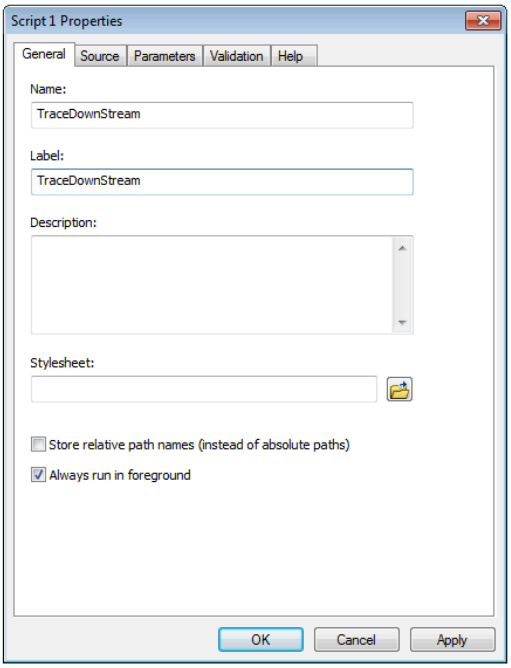
Now that we have some python code that traces a path downstream of any location, we can add some ArcGIS inputs so that it can be used easily.

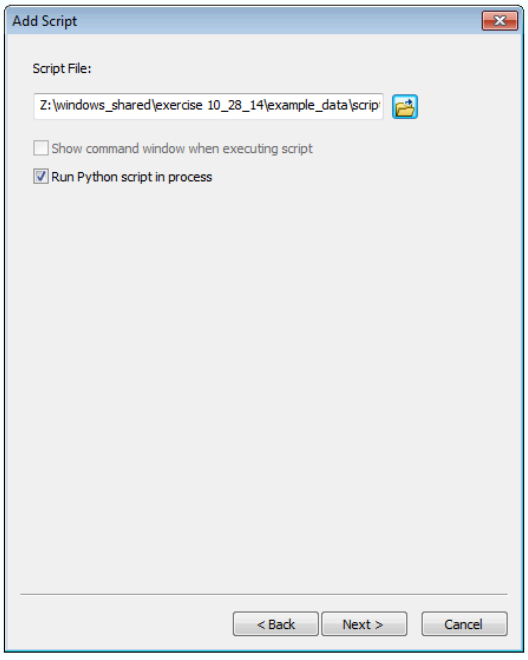
First create a symbolic layer, which will be used in the next step, to assign a theme to one of our inputs. This will also allow us to incorporate an interactive point input selection feature. To do this, right click inside ArcCatalog and select **New -> Shapefile.** Set a name for this file (e.g. my_point.shp) and set the feature type to **Point.** Change the symbology of this point however you would like. Lastly, right click on the my_point.shp in the Table of Contents and select **Save as Layer File**.



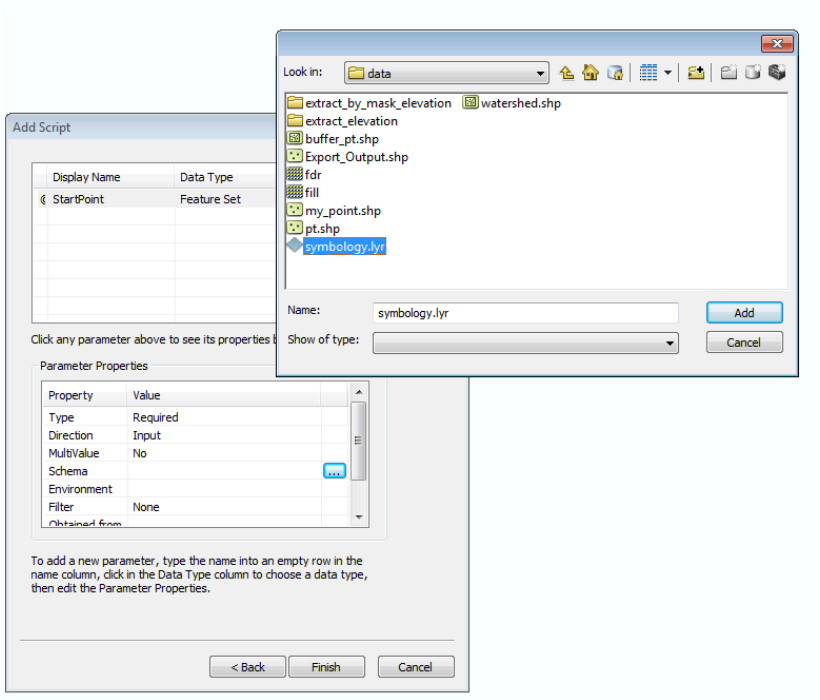


Now lets add our new script to the ArcGIS toolbox, so that we can run it like any other tool. Right click on your toolbox (e.g. Exercise5) and select **Add -> Script**. Give it a name and a label, then select next. Specify the location of the python file.

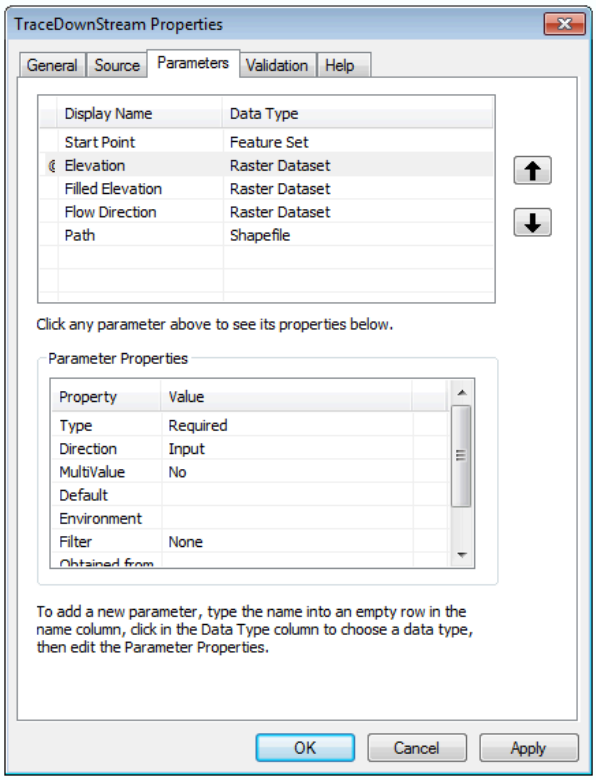




Now lets add some input parameters. The first input parameter will be the start point of the trace operation. Specify a **Display Name** (such as StartPoint) and set the datatype to **FeatureSet**. Next select the **Schema** property and set its value to the symbology layer that we created in the previous step. (e.g. symbology.lyr)



Lets also add parameters for **Elevation (input)**, **Fill (output), Flow Direction (output), and Path (output).** Make sure that the direction parameter for the last three are set to **Output.**



Now we need to add some code to our python script to utilize these parameters. We use the **arcpy.GetParameter(index)** function to grab user inputs from the ArcGIS UI. The following snippet gets the first parameter (i.e. Start Point) as a feature set, and extracts the (x,y) coordinates. This code should be placed directly under the **move_to_next_pixel(fdr, row, col)** function.

```
fs = arcpy.GetParameter(0)
if fs == '#' or not fs:
    fs = "in_memory\\{87AF799A-1608-483B-9022-3AA58EFEF329}" # provide a default value if unspecified

# create feature set
f = arcpy.FeatureSet(fs)

# parse out the geometry
geom = json.loads(f.JSON)['features'][0]['geometry']
pnt = arcpy.Point(geom['x'], geom['y'])
arcpy.AddMessage('Selected Point = (%5.3f,%5.3f)'% (geom['x'], geom['y']))
```

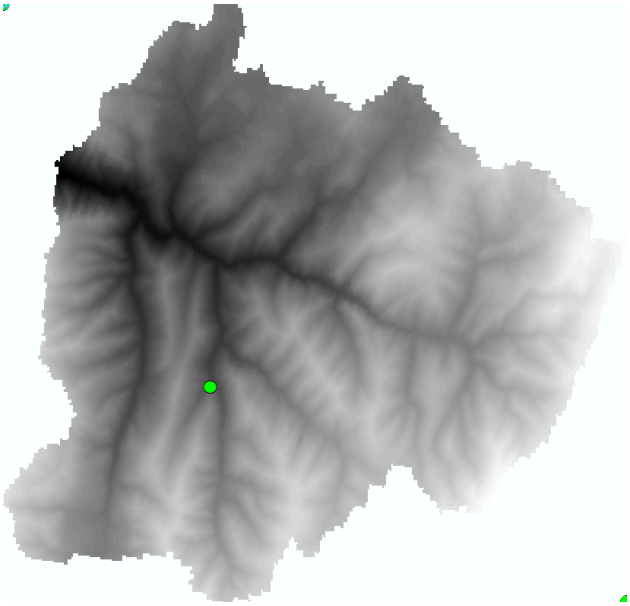
Next, lets add some code to get the rest of our inputs and outputs:

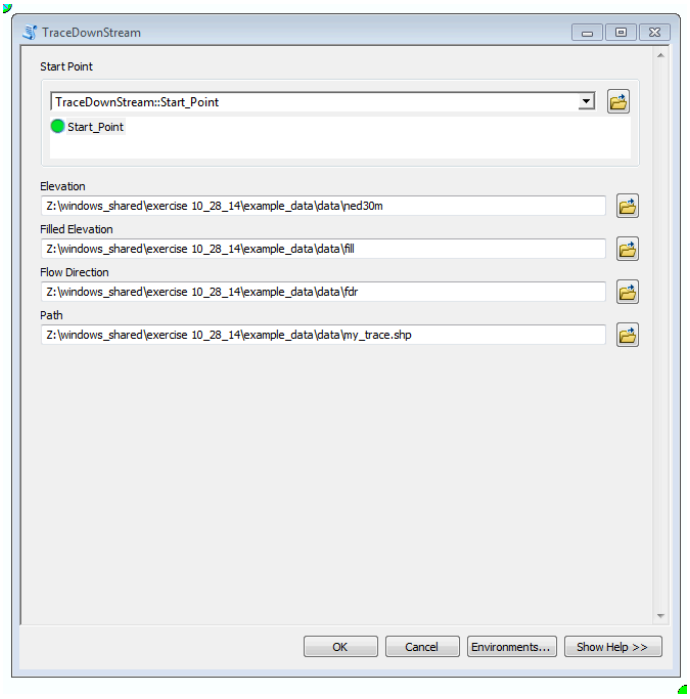
```
# get elevation input
elevation = arcpy.GetParameterAsText(1)

# get output fill path
fill_outpath = arcpy.GetParameterAsText(2)

# get output fdr path
fdr_outpath = arcpy.GetParameterAsText(3)

# get output trace path
trace_outpath = arcpy.GetParameterAsText(4)
```

Since we are getting these parameters from ArcGIS, we need to remove our old hardcoded paths. We should also add some messages, since our print statements will not appear anywhere. Add or remove the following lines in your script.

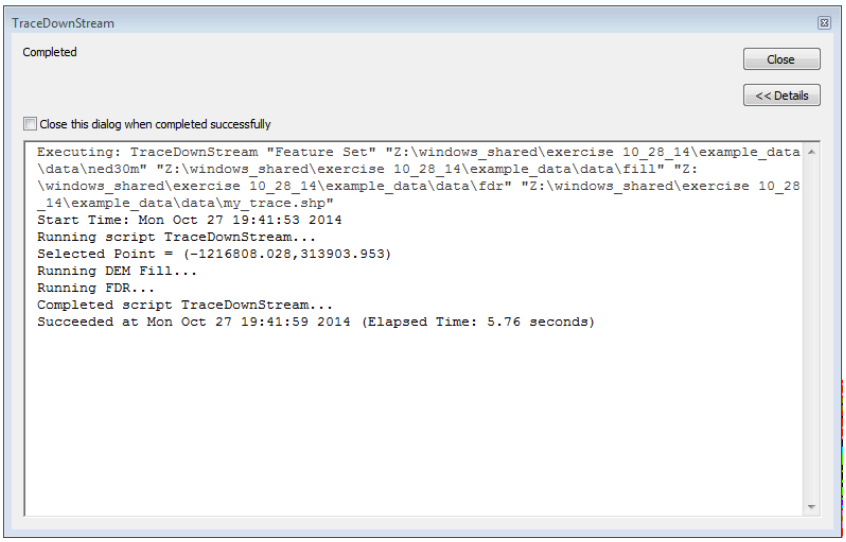
```
import os

...

# Local variables:
# ned30m = "Z:/windows_shared/exercise 10_28_14/example_data/elevation/zipfolder/ned30m"
# fill_outpath = "Z:/windows_shared/exercise 10_28_14/example_data/fill"
# fdr_outpath = "Z:/windows_shared/exercise 10_28_14/example_data/fdr"

# create a point object
# my_x = -1216071.141
# my_y = 307660.098
# pnt = arcpy.Point(my_x, my_y)

# Process: Fill
arcpy.AddMessage('Running DEM Fill...')
# outFill = Fill(ned30m, "")
outFill = Fill(elevation, "")
outFill.save(fill_outpath)

# Process: FDR
arcpy.AddMessage('Running FDR...')
outFlowDirection = FlowDirection(outFill, "NORMAL")
outFlowDirection.save(fdr_outpath)

...

#arcpy.CreateFeatureclass_management('.','path.shp', "POLYLINE")
directory_path = os.path.dirname(trace_outpath)
file_path = os.path.basename(trace_outpath)
arcpy.CreateFeatureclass_management(directory_path,file_path, "POLYLINE")

#cursor = arcpy.InsertCursor('path.shp')
cursor = arcpy.InsertCursor(trace_outpath)
```

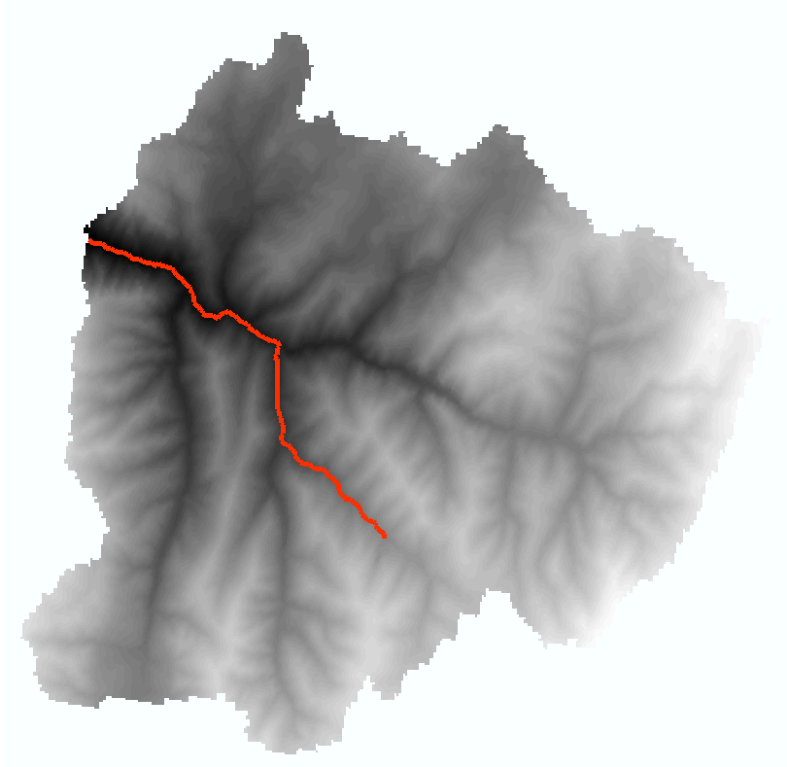
With these changes to our python script, we should be able to successfully run our tool from the ArcGIS toolbox.





Note that our output messages appear in the standard ArcGIS output dialog.





Homework Questions

1. What is the value of the flow direction array at location (134, 289)? Turn in your result along with the line of code that gave you this answer.

2. What is the bounding box of the Flow direction array, (i.e. MinX, MinY, MaxX, MaxY)? Turn in your result along with the line of code that gave you this answer.

3. Write a statement to select the maximum and minimum values of the fill raster. Turn in your result along with the line of code that you used.

4. Explain how the **move_to_next_pixel** function works. Use examples.

5. Modify the **while** loop in this code so that it will terminate at a user defined distance from the input point. For instance, a 1000 meter radius from the input point.

6. How could you modify this code to determine the longest flow path in the entire watershed?

7. Modify the code to provide the min, max, and average slope along the trace path.

8. Explain how you would modify the code to operate on a list of points instead of just one.