Exercise 3: ArcPy cursors

The Cost Distance (Spatial Analyst) tools can be used to determine the least-cost path from a source point to a destination point. This least-cost path actually traverses across a cost raster and finds the cheapest route to travel between the source and destination according to the units defined in the raster (for example, if using a slope cost raster, areas with high slope have a high cost and flat areas, a low cost).

Using Python scripting, a workflow has been created where this least-cost analysis is performed between all source and destination points in two feature classes; we want to create an output table that shows the cost distance for all source\destination pairs. If you are familiar with least-cost path analysis, you know that 2-3 tools are required to generate a path between a single source and a single destination – performing this operation for even 10 source points and 10 destination points would be extremely time consuming. Since the Cost Distance tools themselves do not support calculating the cost distance from **all** source points to **all** destination points, scripting is a necessary piece to achieving these results.

The script is almost finished! The cost path workflow is set up, and currently the output of the script is a Python array (a list of lists).

```
[[0, 0, 53959.921875],
 [0, 1, 47178.73828125],
 [1, 0, 81851.15625],
 [1, 1, 39790.34375]]
```

Each list item in the array corresponds to one source\destination pair and the resulting cost distance for traveling along the least-cost path between the points. Since this Python array is not suited to further analysis with other ArcGIS geoprocessing tools, we want to write this output to a standard table. Using the ArcPy cursor functions, write these values to a table so further analysis can be performed in ArcGIS.

Examine data

- Navigate to folder …Python_ArcGIS\exercises\exercise3_cursors
- Double-click CostDistanceMatrix.mxd to open it in ArcMap, and examine the data layers that are available in the table of contents. The Trailheads layer will be the source points, the Campsites layer will be the destinations, and the Slope layer will be the cost raster.
- The feature classes needed to complete this exercise are stored at …Python_ArcGIS\exercises\exercise3_cursors\data
- Close ArcMap

Setting up the script

- Open PythonWin
- File>Open to the folder …Python_ArcGIS\exercises\exercise3_cursors\scripts
- Open the Python script CostDistanceMatrix.py
- The script file starts the normal way: some comments and then imports of system modules
- You may notice that this script is structured in a more modular way – the script actually starts at the bottom in the "__main__" function, where inputs and outputs are defined, and the first function CostDistanceMatrix is called.

Aside from this different structure, this script has some other interesting components. NOTE: in PythonWin press the keys CTRL+G to open a dialog box where you can enter a line number to jump to.

- Line **28**: Since the script relies on Spatial Analyst extension tools, it is necessary to check if that extension is available. If it is available, it must be checked out. If it is not available, an error is delivered and the script is exited using the sys.exit() function.
- Line **43**: The Describe function is used to retrieve the names of the ObjectID and Shape fields for the source and destination feature classes. Using this function instead of hard coding field names makes the script more portable and guarantees that it will work with data that may have differently named fields.
- Line **53**: Here is the first use of an ArcPy cursor, SearchCursor. In subsequent lines, this cursor is used to go through each feature in the sources feature class.
- Line **58**: The getValue function is used on a cursor row to retrieve a field value. This line gets the ObjectID of the row; the next line gets the actual point geometry of the row.
- Line **72**: The point geometry from the cursor row is used in a geoprocessing tool. Tools that accept feature class input will typically also accept these geometry objects.
- Line **82**: Another SearchCursor is opened, this time on the Cost Path raster. Field values from a raster can be retrieved in the same way as feature class field values.

Creating an output table

Line 38 is where the function that remains to be written is called. By this point, the Python array containing the output values has been created, now we just need to write these values to a table.

- Go to Line 106. The function CreateOutput is already set up; it just needs to be filled in.
- Since no output table even exists yet (we only have a path to the desired output), first use the CreateTable_management tool to create a blank table
  - The CreateTable tool has two required parameters: output location and output name
  - You can use the os.path.dirname(outtable) and os.path.basename(outtable) commands to split the output table path into a location and name that can be used in these parameters

```
#create output table
arcpy.CreateTable_management(os.path.dirname(outtable), os.path.basename(outtable))
```

- This table is totally blank – no fields, no records, etc. Use the AddField_management tool to add three new fields

```
#add fields to output table
arcpy.AddField_management(outtable, "IN_FID", "LONG")
arcpy.AddField_management(outtable, "NEAR_FID", "LONG")
arcpy.AddField_management(outtable, "COST", "DOUBLE")
```

- Finally, use InsertCursor to fill in the output table with records from the Python array
  - Each list inside the main list 'matrix' represents one row in the output table. Iterate through these lists and retrieve values to fill in the output table.

```
#write field values to output
icur = arcpy.InsertCursor(outtable)
for arow in matrix:
        row = icur.newRow()
        row.setValue("IN_FID", arow[0])
        row.setValue("NEAR_FID", arow[1])
        row.setValue("COST", arow[2])
        icur.insertRow(row)
```

- Press the Check button to check the script for syntax errors like indentation mistakes or forgotten colons

- Press the Run button and run the full script

Results

This script provides an excellent example of the utility of Python scripting in ArcGIS – if there isn't a tool that does exactly what you want, build a custom workflow that will produce those desired results. The successful workflow and creation of an output table should look like below.

CostDistanceMatrix

IN_FID	NEAR_FID	COST
0	0	53959.921875
0	1	47178.738281
1	0	81851.15625
1	1	39790.34375

Instead of only extracting the cost distance values from the cost paths, you could also save the path rasters and add them to the map.