**Name: Roy Thorsby**

# LAB 1: Understanding the GIS Environment, Referencing Variables and Calling Functions

Briefly explain what each of the following functions does. Write out the exact syntax that *YOU* entered into the command line to execute the following functions (i.e. the actual variables you created and used).

(NOTE: all syntax assumes that arcpy modules has been called beforehand via import arcpy)

1. **workspace.**

Purpose: Defines a location for the storage, reading and writing of files, inputs and outputs when working with commands and tools. Essentially points to where all your files will go.

Function Syntax:

#declare file pathway of workspace, avoid using escape characters

>>> env.workspace = "E:\\Lab\_1\\Data"

1. **extent.**

Purpose: Within the environments function of arcpy, this prints the extent of a map object, represented as the coordinates of the lower left and upper right corners of a rectangle in map units.

Function Syntax:

# set the extent environment to a target feature

>>> env.extent = "E:\\Lab\_1\\Data\\MOTornadoTracks.shp"

# print the extent to check results

>>> print env.extent

# results shown as coordinates…

44034.9909450899 -315258.096384846 701281.696752604 292872.282081833 NaN NaN NaN NaN

1. **outputCoordinateSystem.**

Purpose: Within env (arcpy), this will create an output geodataset with a specified coordinate system. Changes the geoprocessing environment settings to match a coordinate system of a file, object or reference.

Function Syntax:

#set the coordinate system of outputs to that of a stored .prj file

>>> arcpy.env.outputCoordinateSystem = "E:\\Lab\_1\\Data\\MOTornadoTracks.prj"

# print results to verify..

>>> print arcpy.env.outputCoordinateSystem

<geoprocessing spatial reference object object at 0x35A57EC0>

1. **Rename.**

Purpose: There are many different tools to rename everything. This particular function is within the data management tool section. The purpose is to change the name of a dataset such as a raster, table, or shapefile.

Function Syntax:

# since workspace is already defined, declare variables or an input dataset and a renamed output dataset

# declare variable for input dataset

>>> inputFC = "E:\\Lab\_1\\Data\\MOTornadoPaths.shp"

# declare variable for output dataset with new name

>>> outputFC = "E:\\Lab\_1\\Data\\MOTornadoPaths2.shp"

# call rename function from arcpy to activate renaming process

>>> arcpy.Rename\_management(inputFC,outputFC)

# returns new name as result…

<Result 'E:\\Lab\_1\\Data\\MOTornadoPaths2.shp'>

1. **FeatureToPoint.**

Purpose: Within the data management toolset, this function creates a feature class consisting of points that are constructed based on the representative locations of input features.

Function Syntax:

#call the FeatureToPoint function in arcpy and specify its parameters…

>>>arcpy.FeatureToPoint\_management("MOTornadoPaths2","E:\\Lab\_1\\Data\\MOTornadoCentroid.shp","CENTROID")

# returns the resulting output feature as a result…

<Result 'E:\\Lab\_1\\Data\\MOTornadoCentroid.shp'>

1. **FeatureVerticesToPoints.**

Purpose: Within the data management toolset, this function creates a feature class containing points generated from specified vertices or locations of the input features.

Function Syntax:

# call FeatureVerticesToPoints from arcpy and specify arguments carefully..

>>>arcpy.FeatureVerticesToPoints\_management("MOTornadoPaths2","E:\\Lab\_1\\Data\\MOTornadoEndPoints.shp","BOTH\_ENDS")

# returns output file as a result

<Result 'E:\\Lab\_1\\Data\\MOTornadoEndPoints.shp'>

1. **Densify.**

Purpose: Within the editing tool section, this function adds vertices along line or polygon features, thus densifying objects by a set distance parameter.

Function Syntax:

# call Densify\_edit from the arcpy function and enter arguments

>>> arcpy.Densify\_edit("MOTornadoPaths2","DISTANCE","1000 meters")

<Result 'MOTornadoPaths2'>

1. **FeatureVerticesToPoints.**

Purpose: Within the data management tool section, this tool creates a feature class made of points constructed from vertices or locations specified from an input feature or features.

Function Syntax:

# call FeatureVerticesToPoints\_management from arcpy and enter arguments

>>>arcpy.FeatureVerticesToPoints\_management("MOTornadoPaths2","E:\\Lab\_1\\Data\\MOTornadoVertices.shp","ALL")

#successful function returns new vertices shapefile as a result

<Result 'E:\\Lab\_1\\Data\\MOTornadoVertices.shp'>

1. **Describe.**

Purpose: The purpose of this function is to describe a feature based on a specified property. The description is returned as an object such as the type of data, fields, file name, extension, file path..etc.

Function Syntax:

# call the describe function from arcpy but as a declared variable in which the returned ‘description’ or describe object will be stored. Note that you must specify the property you want described following a period

>>> desc = arcpy.Describe("MOTornadoEndPoints").Extent

#printing the variable will show a successful result, the describe object being the extent of the feature

>>> print desc

44034.9909450899 -315258.096384846 701281.696752604 292872.282081833 NaN NaN NaN NaN

1. **CreateFishnet.**

Purpose: Basically, this tool creates a grid or fishnet of square to rectangular cells of a specified size. This fishnet can be sized over a set of features for grid or cell based analysis of feature data within an area.

Function Syntax:

# CreateFishnet requires several parameters to specify its dimensions and placement

# the fishnet needs an origin coord., y and x axis coord, and a cell width and height

# since we want our grid to encompass the area of our current dataset (tornadoes), we must create variables to use as fishnet parameters based on the orientation and location of our tornado endpoint data

# start by creating the maximum and minimum X andY locations based on the four compass directions

>>> West = arcpy.Describe("MOTornadoEndPoints").Extent.XMin

>>> South = arcpy.Describe("MOTornadoEndPoints").Extent.YMin

>>> East = arcpy.Describe("MOTornadoEndPoints").Extent.XMax

>>> North = arcpy.Describe("MOTornadoEndPoints").Extent.YMax

# concatenate the X and Y max/min coordinates together with “+” in python to make the final coordinates that will be the variables for our fishnet extents

# since this is string data, str() may have to be used for compatibility and syntax

# note that the minimum coordinates must differ from the origin coordinate, so we create an XYmin as an origin and an XYminmod offset by 10 for an extent parameter

>>> XYmin = str(West) + " " + str(South)

>>> XYminmod = str(West) + " " + str(South + 10)

>>> XYmax = str(East) + " " + str(North)

# finally all our parameters are stored in appropriate variables, now call on the arcpy fishnet function and enter in parameters with 5000 unit spacing on grid

# note that 0 is input to tell arcpy to set the parameter as a default

# note that “#” is input to tell arcpy to ignore that parameter or variable in the function

>>>arcpy.CreateFishnet\_management("E:\\Lab\_1\\Data\\5000mFN.shp",XYmin,XYminmod,5000,5000,0,0,XYmax,"#","#","POLYGON")

# successful function output will return a shapefile result

<Result 'E:\\Lab\_1\\Data\\5000mFN.shp'>

1. **SpatialJoin.**

Purpose: The purpose of this function is to join attributes from one feature to another based on a spatial relationship. The target features and the joined attributes from the join features are written to the output feature class

Function Syntax:

#before using the spatial join function. the fields containing the attribute must be mapped for the target features and the join features. I still have little idea how this process works on an intuitive level

# first, start field mapping by creating the fieldmappings object

# note that fieldmappings is a variable or object name, very different from the FieldMappings() function

# FieldMappings object is a collection of FieldMap objects, and it is used as the parameter value for tools that perform field mapping. FieldMappings() has its own list of parameters while FieldMap() is a separate function that can be used with FieldMap datatypes/objects later on??

>>> fieldmappings = arcpy.FieldMappings()

# Initialize the FieldMap objects by adding the input feature classes or tables that are to be combined.

# The FieldMappings object is initialized using the addTable method to enter each input.

# modify the default FieldMappings object by creating two new FieldMap objects, populating their properties, and adding them to the FieldMappings object

>>> fieldmappings.addTable("E:\\Lab\_1\\Data\\2000mFN.shp")

>>> fieldmappings.addTable("E:\\Lab\_1\\Data\\MOTornadoEndPoints.shp")

# Find a field map within the field mappings by name using the findFieldMapIndex parameter within the FieldMappings() function, declared as the fieldmappings variable/object

>>> FSCALEFieldIndex = fieldmappings.findFieldMapIndex("FSCALE")

# Now that the field of interest has been found within the field map, use the getFieldMap function of FieldMappings() to retrieve the specified field map from the index of field mappings created previously

>>> fieldmap = fieldmappings.getFieldMap(FSCALEFieldIndex)

# use outputField within FieldMap() to set the properties of the field map object or objects, works with the data type field

>>> field = fieldmap.outputField

>>> field.name = "meanFscale"

>>> field.aliasName = "meanFscale"

>>> fieldmap.outputField = field

>>> fieldmap.mergeRule = "mean"

>>> fieldmappings.replaceFieldMap(FSCALEFieldIndex,fieldmap)

# finally, execute the spatial join function

>>>arcpy.SpatialJoin\_analysis("2000mFN","MOTornadoEndPoints","TornadoEventSummary.shp","JOIN\_ONE\_TO\_ONE","KEEP\_ALL",fieldmappings,"INTERSECT")

# success returns a shapefile result

<Result 'E:\\Lab\_1\\Data\\TornadoEventSummary.shp'>

1. **MapDocument.**

Purpose: The purpose of this tool is to identify the .mxd map document of interest and provide access to its properties, objects and methods. This step is usually required before proceeding with other arcpy.mapping functions

Function Syntax:

# declare a variable for accessing the current map document using arcpy.mapping.MapDocument

# now with the mapdocument pointing as an object to our mxd file, we can save within python using the Mymxd.save command

>>> Mymxd = arcpy.mapping.MapDocument("E:\\Lab\_1\\Thorsby\_Lab1.mxd")

1. **save.**

Purpose: This method or function within arcpy.mapping saves an .mxd map document

Function Syntax:

# note: this function will not work if the map document is currently open in ArcMap

>>> Mymxd.save("Thorsby\_Lab1.mxd ")

# however one can save a copy instead

>>> Mymxd.saveACopy("bkupmxd.mxd")