Tutorial 2-3 Coding for multiple geoprocessing tasks

Combining geoprocessing tasks with features, such as decision-making logic and feature cursors, allows programmers to make sophisticated scripts.

Learning objectives

- Using cursors and for statements
- Making a feature layer
- Using decision-making logic

Preparation

Research the following topics in ArcGIS for Desktop Help:

- "Accessing data using cursors"
- "Make feature layer (Data Management)"
- "Buffer (Analysis)"
- "Select Layer By Location (Data Management)"
- "An overview of the Layers and Table Views toolset"

Introduction

Running the geoprocessing tools in ArcMap and copying a code snippet is an effective way to get the tool syntax with all the parameters set, but there are still some issues that need to be fixed. That method also will not handle decision-making processes such as if-elif routines. In this tutorial, you will explore another method of developing the tool syntax by researching the script examples in ArcGIS for Desktop Help.

Each tool in the ArcGIS environment and each function and class in ArcPy have a well-documented Help page. This page provides a description of the tool and its parameters, and it typically includes two Python scripting examples—a simple case and a complex case. The simple example will probably make sense to you now, and as you develop your Python skills, the more advanced examples will as well. In fact, you may pick up some good coding techniques from the more complex examples, which you can also use in future scripts.

Scenario

A gas well drilling company has made an application to drill several wells in Oleander. To perform this drilling, the company must notify and get a signed lease document from all the property owners that are located over the planned drill paths. The city's Engineering division has asked that you generate a set of lists of property owners for each drill path. Before each path is drilled, the city will hold a public meeting with those homeowners and the drilling company to work out any issues that might exist. Because you do not know when each well will go online, you will create a separate mailing list for each well path and have them on hand for use when they are needed.

To create the list, buffer the well path and select the properties that intersect the buffer. Different well path lengths require different buffer widths—the longer the path, the wider the buffer. The distances are as follows:

- For well paths less than 3,000 feet, the buffer width is 75 feet.
- For well paths over 3,000 feet but less than 4,000 feet, the buffer width is 175 feet.
- For well paths 4,000 feet and longer, the buffer width is 300 feet.

Data

The data includes the well paths for nine proposed well projects. Also in the map document is the parcel data with a field named Prop_Add, which contains the property address necessary for the mailing list. Start with the most important step—writing the pseudo code.

SCRIPTING TECHNIQUES

Two new techniques are introduced in this tutorial. The first technique uses a cursor to access the rows in the feature class's attribute table or in a stand-alone table. Three types of cursors exist: search, which returns read-only values to the script; insert, which allows you to insert new rows into a table; and update, which allows you to change and delete rows in a table.

Start by defining a cursor object. This object uses one of the cursor commands from the data access module in ArcPy. The cursors have two required arguments, which are the table name and the fields from the table to use in the cursor. A single field or a list object that has many field names can be used in the cursor. The fields are indexed in the cursor in the order in which you list them. For example, index 0 would be the first field in the list, and index 1 would be the second field. The code shown in the graphic defines one of each type of cursor using either a single field or a list of fields.

```
exampleFC = r"C:\EsriPress\GISTPython\Data\OleanderOwnership.gdb\Elm_Fork_Addition"
# Define a search cursor with a single field
searchCur = arcpy.da.SearchCursor(exampleFC,["UseCode"])
# Since only one field is specified, index 0 is the UseCode field

# Define an insert cursor with two fields
insertCur = arcpy.da.InsertCursor(exampleFC,["Prop_Des_1","Prop_Des_2"])
# Field index 0 is Prop_Des_1, and field index 1 is Prop_Des_2

# Define an update cursor with a list of fields
fieldList = ["Prefix","StName","Suffix","SuffDir"]
updateCur = arcpy.da.UpdateCursor(exampleFC,fieldList)
# Field indexes will be 0, 1, 2, and 3 in the order of the fields in the list
```

As an option, the cursor could have a query statement so that only a subset of the rows is accessed.

Once the cursor is created, you can use it to move through the table one row at a time, always moving forward. The for statement has two parameters, the name of the object representing the current row and the cursor. Following these parameters, variables are added to hold the field values from the current row. The code shown in the graphic sets up a variable for each of the four values from the preceding update cursor example.

```
# Use a for statement to go through all the rows
for currentRow in updateCur:
    # Store the field values for the current row in variables
    stPrefix = currentRow[0]
    stName = currentRow[1]
    stSuffix = currentRow[2]
    stSuffDir = currentRow[3]
```

When all your processing is done, and before the script ends, delete the current row object and the cursor object, as shown in the following graphic. Deleting these objects will remove the file locks on the feature class or table you used.

```
# Delete the objects for the current row and the cursor
del currentRow
del updateCur
```

This step is not needed for a search cursor because that type of cursor does not lock the file being accessed.

The other new technique in this tutorial is the use of feature layers. The advantage of using a feature layer instead of a feature class is that the feature layer is a temporary copy of the data that exists only in memory. Changes can be made to the items in a feature layer and the data structure itself without affecting the source file. The changes will not persist after the script ends unless they are explicitly saved to the script. The MakeFeatureLayer tool also allows you to add a selection clause to the process that lets you work with a subset of the data. For instance, the vacant property of Oleander could be put in a separate layer file with the MakeFeatureLayer tool and an optional selection clause of "UseCode = 'VAC.'"

Code multiple geoprocessing tasks

1. **Write the pseudo code for this project. Include tool references and notations of parameters and conditions that you must set.**

   Make sure that only one feature is selected, and that you have the correct feature length to set the buffer. A completed version of the pseudo code is included at the end of this tutorial that you can use for comparison to your own.

2. **Start ArcMap, and open the map document Tutorial 2-3. Also, start your IDE, and create a new Python script named WellNotification.py.**

You can use the basic template of a Python script from tutorial 2-2, but in this tutorial, add some additional environment settings. In tutorial 2-2, code was added to the script to delete temporary feature classes so that they would not become a permanent part of your data. This time, use a scratch workspace to hold the temporary outputs rather than store them in your permanent workspace. To avoid an error, set the geoprocessing environment to allow existing data to be overwritten.

3. **In your IDE, set the workspace to C:\EsriPress\GISTPython\Data\City of Oleander.gdb\Well_Data. Add the environment setting to overwrite the output of geoprocessing tasks, if it exists. Compare your code with the code shown:**

```
# Set up the environment
env.workspace = r"C:\EsriPress\GISTPython\Data\City of Oleander.gdb\Well_Data"
env.overwriteOutput = True
```

4. **Create a variable to hold the name of the feature class that contains the first well path, as shown:**

```
# Create a variable with the name of the subject feature class
fcName = "BC_South_3H_Path"
```

Now that the feature class is known, determine the distance to use in the buffering. The script should check the Shape_Length value for each drill path and set up a condition statement to determine the buffer distance.

To get the field value, use a cursor, which allows you to go through each row in the table for the feature class and get the field values, one by one. For these datasets, each feature class has only one feature, so the first returned value can be used to determine the buffer path.

Of the several types of cursors, use a search cursor for this task. Research SearchCursor in ArcGIS for Desktop Help or any other ArcPy reference you have available. Next, set up the cursor to find the Shape_Length field, and then store that value in a variable.

5. **Write the code to create a search cursor named wellCursor. Add the statements to store the value of Shape_Length in a variable named drillLength, as shown:**

```
# The buffer distance is dependent on the length of the drill path
# Check to see how long the path is
wellCursor = arcpy.da.SearchCursor(fcName,["Shape_Length"])
for row in wellCursor:
    drillLength = row[0]
```

Use the length of the feature to determine the buffer distance. The scenario at the start of this tutorial describes the conditions for each distance.

6. **Set up an if-elif-else statement to determine the correct buffer distance, and create a variable to store it named wellBuffDist, as shown:**

```
# < 3000 feet uses a buffer of 75'
# >= 3000 feet and < 4000 feet uses a buffer of 175'
# >= 4000 feet uses a buffer of 300'
if drillLength < 3000:
    wellBuffDist = 75
elif drillLength >= 3000 and drillLength < 4000:
    wellBuffDist = 175
else:
    wellBuffDist = 300
```

Next, buffer the input feature class by the determined well buffering distance. Store the output in a file named **SelectionBuffer,** and store it in a separate workspace, **C:\EsriPress\GISTPython\MyExercises\Scratch\TemporaryStorage.gdb.**

7. **Find the tool documentation for buffer, and use the examples shown to set up the correct buffer statement.**

```
# Perform the buffering
# Buffer_analysis (in_features, out_feature_class, buffer_distance_or_field,
#     {line_side}, {line_end_type}, {dissolve_option}, {dissolve_field})
arcpy.Buffer_analysis(fcName, \
r"C:\EsriPress\GISTPython\MyExercises\Scratch\TemporaryStorage.gdb\SelectionBuffer", \
wellBuffDist)
```

With the buffer completed, move on to the selection process. Use the new buffer to select the parcels that intersect it. If you were doing this manually in ArcMap, you would use the Select By Location tool from the ArcMap Selection menu. A counterpart is available in ArcPy named Select Layer By Location, but the ArcPy selection tool will act only on a feature layer, not on a feature class. You must write code to make the input feature class a feature layer, and then add code to make the selection.

8. **Research the Make Feature Layer and Select Layer By Location tools, and use the code samples to determine the code for this project, as shown in the graphic. (Hint: use the full path name for the Parcels input layer because it is not coming from the default workspace you set.)**

```
# Use buffer to select the parcels
# Make a feature layer to temporarily hold the input data
arcpy.MakeFeatureLayer_management(r"C:\EsriPress\GISTPython\Data\City of Oleander.gdb\Parcels", \
"Parcels_Lyr")

# Use the feature layer in the slection process
arcpy.SelectLayerByLocation_management("Parcels_lyr", "INTERSECT",\
r"C:\EsriPress\GISTPython\MyExercises\Scratch\TemporaryStorage.gdb\SelectionBuffer")
```

Now that you have the features within the buffer selected, the last step is to write them out to a new table that can be used with mail-merge software. Note that this should not be a feature class. A search of the Help files produces two tools that look like they might work for this step: Table To Table and Copy Rows. Research these tools, and determine which one would work best.

9. **Using your research, add the code to write the selected features to a new table formatted as the input feature class name with the word MailList appended at the end, as shown. When the code is completed, save the script.**

```
# Copy the selected features to a new table
arcpy.CopyRows_management("Parcels_lyr", r"C:\EsriPress\GISTPython\Data\\" \
+ fcName + "_MailList.dbf")
```

Note the double backslash (\\) at the end of the folder string. Because this is pointing to a folder and not a file, Python requires the additional formatting character—otherwise, no backslash would appear between the completed file path and the file name.

10. **To test the script, run it and select one of the drill path feature classes from the City of Oleander.gdb\Well_Data feature dataset.**

11. **Add the new database table to your ArcMap document and open it. It contains all the fields from the Parcels layer, including the one you need, as shown:**

Table

BC_South_3H_Path_MailList

OID	OBJECTID	Prop_Des_1	Prop_Des_2	Acreage	DU	PlatStatus	UseCode	PIDN	Prefix
0	5148	BEAR CREEK BEND ADDITION	BLK A LOT 24	0.174968	1	1	A1	1899 A 24	
1	5217	MIDWAY PK	BLK 18 LOT 10	0.996592	0	1	F1	25940 18 10	W
2	6652	MIDWAY SQUARE ADDITION	BLK A LOT 18	0.163653	1	1	A5	25975 A 18	
3	6658	MIDWAY SQUARE ADDITION	BLK A LOT 17	0.154063	1	1	A5	25975 A 17	
4	7196	WOODCREEK	BLK A LOT 43	0.102241	1	1	A5	47485 A 43	
5	7204	HARWOOD COURTS	BLK E LOT 4	0.103758	1	1	A5	17402 E 4	
6	7481	BEAR CREEK BEND ADDITION	BLK A LOT 32	0.207402	1	1	A1	1899 A 32	
7	7482	BEAR CREEK BEND ADDITION	BLK A LOT 33	0.142065	1	1	A1	1899 A 33	
8	7483	BEAR CREEK BEND ADDITION	BLK A LOT 34	0.114024	1	1	A1	1899 A 34	
9	7487	BEAR CREEK BEND ADDITION	BLK A LOT 35	0.13864	1	1	A1	1899 A 35	
10	7496	HARWOOD CROSSING	BLK 1 LOT 3	4.31566	0	1	F1	17403 1 3	N
11	6513	MIDWAY SQUARE ADDITION	BLK B LOT 3	0.160455	1	1	A5	25975 B 3	
12	6520	MIDWAY SQUARE ADDITION	BLK B LOT 4	0.162595	1	1	A5	25975 B 4	
13	6741	ARBOR GLEN ADDITION	BLK A LOT 72	0.066816	1	1	A4	796C A 72	
14	6743	ARBOR GLEN ADDITION	BLK A LOT 81	0.092001	1	1	A4	796C A 81	
15	6437	MIDWAY SQUARE ADDITION	BLK A LOT 25	0.152216	1	1	A5	25975 A 25	
16	6631	MIDWAY SQUARE ADDITION	BLK A LOT 19	0.304003	1	1	A5	25975 A 19	
17	6903	WOODCREEK	BLK E LOT 32	0.20849	4	1	B4	47485 E 32	E
18	6376	MIDWAY SQUARE ADDITION	BLK C LOT 3	0.161747	1	1	A5	25975 C 3	
19	6407	MIDWAY SQUARE ADDITION	BLK A LOT 24	0.16022	1	1	A5	25975 A 24	
20	6415	MIDWAY SQUARE ADDITION	BLK A LOT 23	0.167889	1	1	A5	25975 A 23	
21	6439	MIDWAY SQUARE ADDITION	BLK B LOT 1	0.18062	1	1	A5	25975 B 1	
22	6448	MIDWAY SQUARE ADDITION	BLK A LOT 22	0.151065	1	1	A5	25975 A 22	

1 (0 out of 177 Selected)

BC_South_3H_Path_MailList

You can supply this table to the city engineers, and they can use it when necessary to notify the property owners as each well is drilled.

12. **Test your script on one or two of the other drill paths.**

Here's the pseudo code for this task:

```
# Import the modules
# Set up the environment
# Create a variable with the name of the subject feature class
# The buffer distance is dependent on the length of the drill path
# Check to see how long the path is
# < 3000 feet uses a buffer of 75'
# >= 3000 feet and < 4000 feet uses a buffer of 175'
# >= 4000 feet uses a buffer of 300'
# Perform the buffering
# Use buffer to select the parcels
# Make a feature layer to temporarily hold the input data
# Use the feature layer in the selection process
# Copy the selected features to a new table
```