Chapter 2

Writing stand-alone Python scripts

Introduction

Much of the Python programming you create will be stand-alone Python scripts, which you may also bring into ArcGIS as script tools. The scripts will follow a standard format and are written using an integrated development environment (IDE), such as IDLE, which comes with Python; PythonWin, which is popular with many programmers; or PyScripter, which is gaining popularity and is used for the examples in this book. You may use any of these IDEs because the formatting of the scripts and the code you write will be the same, regardless of the IDE used. Please read the documentation of the IDE you choose to learn how to create, run, and save Python programs.

One of the most important things you can do in your code development is to write proper pseudo code. Pseudo code is a plain-text description and diagram of what your code should accomplish. You should note which modules you need to import, which tools you might need to use, the steps necessary to perform the desired operations, which variables you might need to set up and use, and any research notes about the Python code you write. This description is not intended to be code, but it may become documentation in the script for future reference. You wrote a little pseudo code in the first chapter of this book. The pseudo code for these next chapters should be well written and researched and contain detailed descriptions of your process. A simple outline of the pseudo code is found at the end of each tutorial. You should prepare your own pseudo code first, and then compare it with the outline provided.

This chapter reviews some of the basic structure of Python code, but you should use a good Python reference book for more detailed study. Each tutorial is designed to highlight particular techniques to interface your code with ArcGIS. As always, if you find a better or more efficient way to accomplish the same task, by all means, give it a try. Writing and troubleshooting the code is good practice.

Special introduction: Working with Python

Python is similar to other object-oriented programming languages because it deals with functions, classes, and objects. Esri has written a module for Python, named ArcPy, that adds special functions, classes, and objects that deal specifically with ArcGIS and its components. In addition to standard Python code elements, ArcPy also provides access to all the tools available in ArcToolbox. To be successful, learn more about these elements and how to find references to their use. These tool references are easy to find in the materials provided with ArcGIS. The easiest and most straightforward method to access these tools is to search for the tool in ArcGIS for Desktop Help. Another easy way to get a tool reference is to right-click the tool in ArcToolbox, and select Item Description. A third way is to search for the tool in the Search window, and click the link provided with the tool's name. Whichever way you choose, you will be provided with an explanation of the tool's function, the syntax for the tool's parameters, and code samples showing how to use the code in Python scripting.

The simplest code to program uses the ArcGIS tools. Any tool found in ArcToolbox can be brought into a Python script. The basic syntax for using these tools is to call the ArcPy module, add a period, add the tool name and then an underscore, followed by the alias of the toolbox where this tool resides, and then add a set of parentheses containing the parameters necessary to run the tool. For example:

Module.Tool_ToolboxAlias(param1, param2)

This syntax may seem difficult, but code samples are given in the tool reference for each tool. For example, the Buffer tool would be added like this:

```
• arcpy.Buffer_analysis(in_features,out_feature_class,buffer_distance)
```

How do you know things like the toolbox alias and the tool parameters? Simple: look in the tool's reference. The Help for the Buffer tool includes the tool syntax, a description of the parameters, and a couple of sample scripts to show how the tool is used in Python. As good practice, you could copy and paste the tool syntax into your script as a comment so that as you write out the parameters, you

are sure to put them in the correct order. Also, pay close attention to the capitalization. Remember, Python is case sensitive.

Here is the tool reference for Buffer (shown without the optional parameters):

Syntax

Buffer_analysis (in_features, out_feature_class, buffer_distance_or_field, {line_side}, {line_end_type}, {dissolve_option}, {dissolve_field})

Parameter	Explanation	Data Type
in_features	The input point, line, or polygon features to be buffered.	Feature Layer
out_feature_class	The feature class containing the output buffers.	Feature Class
buffer_distance_or_field	The distance around the input features that will be buffered. Distances can be provided as either a value representing a linear distance or as a field from the input features that contains the distance to buffer each feature. If linear units are not specified or are entered as Unknown, the linear unit of the input features' spatial reference is used. When specifying a distance in scripting, if the desired linear unit has two words, like Decimal Degrees, combine the two words into one (for example, '20 DecimalDegrees').	Linear unit; Field

Look up a few commonly used tools in ArcGIS for Desktop Help or some tools in ArcToolbox, such as creating a file geodatabase and performing a union, and note the syntax and code examples provided. If you are not familiar with the exact tool name, use the Search window to find it. Be careful because some tools that you may commonly access from the toolbar when working manually may have different names when used as ArcPy tools. Good examples of these tools include Select By Location on the menu, which becomes Select Layer By Location in ArcPy, and Export from the Table Options menu, which becomes Copy Rows in ArcPy.

In many cases, the tools accessed through ArcPy require the path to files or workspaces. The backslash (\) is a protected character in Python, so system paths that include a backslash must be handled in a certain way. There are three ways to handle these paths: (1) use a double backslash (\\); (2) substitute a forward slash (/); or (3) put a lowercase *r* at the beginning of the character string containing the path. The three examples shown in the graphic would all point to the path correctly in a Python script.

```
  # Use a double backslash
• featureClass1 = "C:\\EsriPress\\GISTPython\\Data\\City of Oleander.gdb\\City_Area"
  # Substitute a forward slash
• featureClass2 = "C:/EsriPress/GISTPython/Data/City of Oleander.gdb/City_Area"
  # Place a lowercase r in front of the string
• featureClass3 = r"C:\EsriPress\GISTPython\Data\City of Oleander.gdb\City_Area"
```

Another programming item provided by ArcPy is functions. Common Python functions include the string and math functions, such as str(x), which casts a numeric variable as a string variable, and round(x,n), which rounds a numeric variable to *n* digits. Many of these functions are used in chapter one. A list of all the ArcPy functions (too long to list here) is available in ArcGIS for Desktop Help. These functions may perform simple tasks, such as adding error or warning messages to your script or determining the current license level.

Functions that are more complex and useful are used more often. The first of these is the Exists() function. This function is used with any dataset object to see if it already exists. It is often necessary to

make sure that the output of a tool does not exist before trying to create it, such as geodatabases, feature datasets, tables, and feature classes. If the output already exists, it may cause your script to malfunction, so it is useful to check for the existence of items first.

Another commonly used function is a list function, and ArcPy has several types of them. List functions are used to create list objects of a variety of items that you will use in your script. All the list functions are shown in the graphic:

ListDatasets
ListDataStoreItems
ListEnvironments
ListFeatureClasses
ListFields
ListFiles
ListIndexes
ListInstallations
ListPrinterNames
ListRasters
ListSpatialReferences
ListTables
ListToolboxes
ListTools
ListTransformations
ListUsers
ListVersions
ListWorkspaces

The result of a list function is a list object, which is used with looping structures to iterate through the list. For example, if you wanted to see if a field named ROW_Width was in a feature class, you would use the ListFields() function to create a list object of the fields, and then

iterate through the list to see if that name were there. As you can see, many items can be included in lists.

One of the most important features of ArcPy is a script's capacity to get input from the user. Without this feature, your scripts would just do the same things over and over. Python includes a function named raw_input() to get user input, but ArcPy includes a special function to do this named GetParameterAsText(). This function is used when you intend to run your script as a script tool in a toolbox. Conversely, a function named SetParameterAsText() returns values from your script tool in the event that you are using the script tool in a model. More about the use of these functions is found in later tutorials as you begin to write more code and use other code samples.

The next concept to master in Python is working with objects. An object is used to hold multiple characteristics or parameters of an element you may be using in your code. Interestingly, objects can be created with simple definition statements in the same way as variables, or objects can be returned from functions.

For example, a feature class is defined as an object in ArcPy. With the simple code shown in the graphic, you can create a feature class object.

```
roadsFeatureClass = r"c:\workspace\Roads_Feature_class"
```

Although the code may look simple, a feature class is a complex object with a variety of characteristics and properties that you can access. For a feature class, these attributes may include the name, file type, path, and extension. The tricky part is accessing these attributes through your Python code, which is accomplished using the Describe() function. The Describe() function outputs a describe object with all the properties available for easy use. For instance, the code in the following graphic creates a feature class object that holds the street centerlines feature class. The Describe() function is used to create a describe object for the feature class, which contains properties of the feature class. The print statement shown will print the name of the feature class, and the .path method will retrieve the path to the folder where the feature class is stored.

```
import arcpy
roadsFeatureClass = r"C:\EsriPress\GISTPython\Data\City of Oleander.gdb\Street_Centerlines"
# roadsFeatureClass = arcpy.GetParameterAsText(0)  # Optional code to prompt for user input
descFC = arcpy.Describe(roadsFeatureClass)
print descFC.baseName
arcpy.env.scratchWorkspace = descFC.path
```

With the file location hard-coded to a specific geodatabase, this may not seem too important, but note the optional code shown that would ask the user to enter a feature class location. A describe

object must be created from the feature class to extract the feature class properties. The code uses .baseName to get the name of the feature class and .path to get the data path, which then is used to set the scratch workspace.

The Describe() function can be used on just about any type of file and dataset brought into ArcGIS. The list in the graphic shows the variety of objects this function will act upon.

Describe Object Properties
ArcInfo Workstation Item Properties
ArcInfo Workstation Table Properties
CAD Drawing Dataset Properties
CAD FeatureClass Properties
Cadastral Fabric Properties
Coverage FeatureClass Properties
Coverage Properties
Dataset Properties
dBASE Table Properties
Editor Tracking Dataset Properties
FeatureClass Properties
File Properties
Folder Properties
GDB FeatureClass Properties
GDB Table Properties
Geometric Network Properties
LAS Dataset Properties
Layer Properties
Map Document Properties
Mosaic Dataset Properties
Network Analyst Layer Properties
Network Dataset Properties
Prj File Properties
Raster Band Properties
Raster Catalog Properties
Raster Dataset Properties
RecordSet and FeatureSet Properties
RelationshipClass Properties
RepresentationClass Properties
Schematic Dataset Properties
Schematic Diagram Properties
Schematic Folder Properties
SDC FeatureClass Properties
Shapefile FeatureClass Properties
Table Properties
TableView Properties
Text File Properties
Tin Properties
Tool Properties
Toolbox Properties
Topology Properties
VPF Coverage Properties
VPF FeatureClass Properties
VPF Table Properties
Workspace Properties

The first item listed (Describe Object Properties) has properties associated with whatever object you are describing, while the other properties are specific to their individual object type. The properties .baseName (the name of the item) and .path (the folder or workspace containing the item) are part of the common object properties of such things as .extension (the file extension if the item is a file) and .children (a list element of items in a workspace or feature dataset). One of the most useful methods is .dataType, which returns the item's data type, such as FeatureClass, Workspace, FeatureDataset, File, or LocalDatabase. The script in the following graphic can be used to print all the describe object properties of any item, and these properties could be used

to check that the correct file type is being used, to store the workspace for data creation, or to capture an item's name.

```
#Import the ArcPy module
import arcpy

# List all describe object properties
# Substitute any item into the Describe command for a list of properties
desc = arcpy.Describe("C:\ESRIPress\GISTPython\Data\City of Oleander.gdb\Planimetrics")

# Print some describe object properties
#
if hasattr(desc, "baseName"):
    print "Base Name: " + desc.baseName
if hasattr(desc, "catalogPath"):
    print "CatalogPath: " + desc.catalogPath
if hasattr(desc, "childrenExpanded"):
    print "Children Expanded: " + str(desc.childrenExpanded)
if hasattr(desc, "dataElementType"):
    print "Data Element Type: " + desc.dataElementType
if hasattr(desc, "dataType"):
    print "DataType:     " + desc.dataType
if hasattr(desc, "extension"):
    print "Extension: " + desc.extension
if hasattr(desc, "file"):
    print "File: " + desc.file
if hasattr(desc, "fullPropsRetrieved"):
    print "Full Properties Retrieved: " + str(desc.fullPropsRetrieved)
if hasattr(desc, "metadataRetrieved"):
    print "Metadata Retrieved: " + str(desc.metadataRetrieved)
if hasattr(desc, "path"):
    print "Path: " + desc.path
if hasattr(desc, "name"):
    print "Name:         " + desc.name


# Examine children and print their name and dataType
#
print "Children:"
for child in desc.children:
    print "\t%s = %s" % (child.name, child.dataType)
```

Each item type will also have a set of describe properties to access properties unique to the particular item, such as Text File Properties and Geometric Network Properties. For instance, to access additional properties of a feature class, you might look at the Dataset Properties, FeatureClass Properties, File Properties, GDB FeatureClass Properties, Layer Properties, and Table Properties to uncover a variety of different properties.

Classes are a more complex element in ArcPy and are used to create objects to hold tool parameters independent of feature classes and workspaces. The long list of these classes is found in ArcGIS for Desktop Help. These classes may be used to set up the parameters for items such as fields, spatial references, and extents before they are applied to existing geometry objects, or these classes may be used to store these parameters after they are extracted from geometry objects.

A common thread in using all these ArcPy features is ArcGIS for Desktop Help, which is beneficial for finding the syntax, usage, and code examples of these features. As stated before, it is good practice to copy and paste information from the Help files into your script as comments to remind you of how an element can be used.

By using combinations of functions, classes, objects, and tools, you will be able to write scripts that will do amazing things with your geographic data.

Tutorial 2-1 Creating describe objects

The ArcPy module provides access to all the geoprocessing tools for tasks you can perform in ArcGIS. These tasks can be done one at a time or used for batch processing with a loop routine.

Learning objectives

- Writing pseudo code
- Using the Describe() function
- Learning ArcPy module basics
- Using Python environment variables
- Using basic error handling

Preparation

Research the following topics in ArcGIS for Desktop Help:

- "A quick tour of Python"
- "Importing ArcPy"
- "Using environment settings in Python"

Introduction

This tutorial demonstrates some of the basic code structure to get your Python scripts to interface with ArcGIS. The main component is the use of the ArcPy module. A module is a special set of tools that you bring into Python to add functionality in extending the basic Python code. In tutorial 1-5, you used a Python module named DateTime, which adds special time handlers. ArcGIS users can import the Esri module ArcPy, which provides access to all the geoprocessing tools and many special features associated with spatial data. For more information about ArcPy, go to the ArcGIS for Desktop Help Contents tab, and navigate to Desktop > Geoprocessing > ArcPy > Introduction.

Scenario

You have received some geographic data in a folder, and you would like to get information about some of its elements. Try your new Python skills and learn more about the layout of a stand-alone Python script by writing a Python script to report the information. Here's the pseudo code for this script:

- Import Python modules.
- Set the workspace environment.
- Check the spatial reference for a feature class in the workspace.
- Print the results to the screen.

Data

In the Data folder of the online data that goes with this book is a file geodatabase named Sample Data, which contains the feature classes shown:

Name	Type
arbordaze2009tents	File Geodatabase Feature Class
complan	File Geodatabase Feature Class
libsprk	File Geodatabase Feature Class
ROW_And_Easements	File Geodatabase Feature Class
sprinklerunit	File Geodatabase Feature Class
ZIPCODES_poly	File Geodatabase Feature Class

SCRIPTING TECHNIQUES

This script and all the scripts that follow start with code to import the ArcPy module and the env class and then to set the workspace environment. Almost every script you write will perform these three tasks at the beginning. Importing the modules is not necessary if the script is run as a tool in ArcMap or ArcCatalog, but it does no harm to include this code. Other environment settings, such as adding the results to the current map document, also are used only if you are working from a script tool in a map document.

In addition, this script uses the Describe() function to get information about a feature class. The syntax is to make a variable equal to arcpy.Describe(feature class name), resulting in a describe object. This syntax can be used in a variety of ways, depending on the information you are extracting. The code shown creates a describe object and then uses that object to extract parameters of the feature class.

```
import arcpy
# Describe the feature class Elm_Fork_Addition, and make a describe object
streetDescObj = arcpy.Describe(r"C:\EsriPress\GISTPython\Data\OleanderOwnership.gdb\Elm_Fork_Addition")
# Get the properties of the feature class from the describe object
fcName = streetDescObj.baseName # Name of feature class
fcPath = streetDescObj.catalogPath # Path to the file
fcShape = streetDescObj.shapeType # Feature type as Point, Polyline, Polygon etc...
fcSpatIndex = streetDescObj.hasSpatialIndex # True/False for the existence of a spatial reference
```

Once the variables are set to the different parameters, they can be used in if statements and other constructs within the script.

Write loop routines

1. **Start your IDE, and create a new file named Tutorial_2-1.py. Add comments at the top of the code block to name and describe your program. Add the pseudo code as comments, which will help when you start entering the code.**

```
#---------------------------------------------------------------------------
# Name:         Tutorial 2-1.py
# Purpose:      Learn the basic format of Python code used in ArcGIS
#
# Author:       David W. Allen, GISP
#
# Created:      10/25/2013
# Copyright:    (c) David 2013
#
#---------------------------------------------------------------------------

#Import Python modules

#Set the workspace environment

#Check the spatial reference for a feature class in the workspace

#Print the results
```

Start by writing the code. The first step is to import the ArcPy module. If this script were to run in ArcGIS, this step would not be needed, but it does no harm to include it in case you want to run the script outside ArcGIS.

The standard format to add the ArcPy module is to type "import arcpy." Arcpy contains several functions, classes, and modules, which supply specialized tools. The env class provides access to the ArcGIS environment setting; the data access (arcpy.da) module provides access to various data access tools and functions; and the mapping (arcpy.mapping) module provides access to the settings and characteristics of map documents. These modules are imported as part of ArcPy, or they can be imported separately by using the format "from arcpy import env."

2. **After the first line of pseudo code, add the statements import arcpy and from arcpy import env, as shown. This code will import the modules necessary to access the ArcGIS tools and the file geodatabase.**

```
  #Import Python modules
* import arcpy
* from arcpy import env

  #Set the workspace environment

  #Check the spatial reference for a feature class in the workspace

  #Print the results
```

Notice that the env class was also loaded, which will help shorten the instructions you will need to write in your code to access the env properties. A few of the more commonly used properties are shown here, but the entire list can be found in ArcGIS for Desktop Help.

Property	Explanation	Data Type
addOutputsToMap (Read and Write)	Set whether tools' resulting output datasets should be added to the application display.	Boolean
referenceScale (Read and Write)	Tools that honor the Reference Scale environment will consider the graphical size and extent of symbolized features as they appear at the reference scale. Learn more about referenceScale	Double
scratchFolder (Read Only)	The Scratch Folder is the location of a folder you can use to write file-based data, such as shapefiles, text files, and layer files. It is a read-only environment that is managed by ArcGIS. Learn more about scratchFolder	String
scratchGDB (Read Only)	The scratch GDB is the location of a file geodatabase you can use to write temporary data. Learn more about scratchGDB	String
scratchWorkspace (Read and Write)	Tools that honor the Scratch Workspace environment setting use the specified location as the default workspace for output datasets. The Scratch Workspace is intended for output data you do not wish to maintain. Learn more about scratchWorkspace	String
workspace (Read and Write)	Tools that honor the Current Workspace environment setting use the workspace specified as the default location for geoprocessing tool inputs and outputs. Learn more about currentWorkspace	String

3. **Add code to set the workspace property to the location where you installed the student data, as shown:**

```
#Set the workspace environment
env.workspace = r"C:\EsriPress\GISTPython\Data\Sample Data.gdb"
# arcpy.env.workspace = r"C:\EsriPress\GISTPython\Data\Sample Data.gdb"
```

Note the second line in this graphic is how you would call the env class if you had not used the from-import statement earlier. Any of the env properties can be set in the same manner, as shown:

```
env.scratchWorkspace = r"C:\EsriPress\GISTPython\Data\City of Oleander.gdb"
env.addOutputsToMap = True
env.referenceScale = '25000'
```

Following the guide of the pseudo code, the next step is to find the spatial reference of a named feature class. Finding the spatial reference involves creating a Python object to contain the properties of the feature class. These objects are created the same way that variables are created.

4. **Because you have already set the workspace, the Python script will know where to look for the feature class you name, so set a variable named fcName to ZIPCODES_poly, as shown:**

```
#Check the spatial reference for a feature class in the workspace
fcName = "ZIPCODES_poly"
```

Once this variable is set, you can access the properties of the feature class using the Describe() function, including the name, feature type, path, file extension, and extent. The table shows a few of these properties, including the presence of a spatial reference.

Property	Explanation	Data Type
canVersion (Read Only)	Indicates whether the dataset can be versioned	Boolean
datasetType (Read Only)	Returns the type of dataset being described • Any • Container • Geo • FeatureDataset • FeatureClass • PlanarGraph • GeometricNetwork • Topology • Text • Table • RelationshipClass • RasterDataset • RasterBand • TIN • CadDrawing • RasterCatalog • Toolbox • Tool • NetworkDataset • Terrain • RepresentationClass • CadastralFabric • SchematicDataset • Locator	String
spatialReference (Read Only)	Returns the SpatialReference object for the dataset	SpatialReference

The syntax for describing a layer is:

```
arcpy.Describe(layername)
```

5. **Write the code to create a describe object named fcNameProperties that references the properties of fcName. Check your code against the code shown:**

```
    #Check the spatial reference for a feature class in the workspace
•   fcName = "complan"
•   fcNameProperties = arcpy.Describe(fcName)
```

By adding the .spatialReference parameter at the end of the describe object, you will gain access to the many spatial reference properties of the layer. These properties include all the technical specifications of the spatial reference, but you are interested in the name. The syntax for getting the name of the spatial reference is:

```
• print describeObject.spatialReference.name
```

6. **Write the code to print the name of the spatial reference of fcNameProperties. If your script matches the one shown, save and run it.**

```
#-------------------------------------------------------------------------------
# Name:         Tutorial 2-1.py
# Purpose:      Learn the basic format of Python code used in ArcGIS
#
# Author:       David W. Allen, GISP
#
# Created:      10/25/2013
# Copyright:    (c) David 2013
#
#-------------------------------------------------------------------------------

#Import Python modules
• import arcpy, sys
• from arcpy import env

#Set the workspace environment
• env.workspace = r"C:\EsriPress\GISTPython\Data\Sample Data.gdb"
# arcpy.env.workspace = r"C:\EsriPress\GISTPython\Data\Sample Data.gdb"

#Check the spatial reference for a feature class in the workspace
• fcName = "ZIPCODES_poly"
• fcNameProperties = arcpy.Describe(fcName)

#Print the results
• print fcNameProperties.spatialReference.name
```

The feature class has the spatial coordinate system as shown:

Python Interpreter

```
NAD_1983_StatePlane_Texas_North_Central_FIPS_4202_Feet
>>>
```

Your turn

Modify the code to examine the feature classes arbordaze2009tents and complan, and determine whether these classes have a spatial reference.

What would happen if you misspelled one of the layer names? The code would have some serious issues with variables that do not exist and a Describe() function that does not work.

7. **Add some error handling to your code with the Python try and except statements. Basically, add** `try:` **at the top of your code and** `except:` **at the bottom. After the except statement, add a print statement to warn the user of an error. The basic syntax is:**

```
try:
    #
    #
    # All your code goes here
    #
    #
except:
    print "Your warning message"
```

Note that the try-except routine requires that you indent all your code, in addition to indenting your warning message at the end.

8. **As shown, add the try-except error handing to your script with the message** `"I couldn't find that feature class!"` **Then type an incorrect file name and run it.**

```
#---------------------------------------------------------------------------
# Name:        Tutorial 2-1.py
# Purpose:     Learn the basic format of Python code used in ArcGIS
#
# Author:      David W. Allen, GISP
#
# Created:     10/25/2013
# Copyright:   (c) David 2013
#
#---------------------------------------------------------------------------

try:
    #Import Python modules
    import arcpy, sys
    from arcpy import env

    #Set the workspace environment
    env.workspace = r"C:\EsriPress\GISTPython\Data\Sample Data.gdb"
    # arcpy.env.workspace = r"C:\EsriPress\GISTPython\Data\Sample Data.gdb"

    #Check the spatial reference for ZIPCODES_poly
    fcName1 = "ZIPCODES_poly"
    fcName1Properties = arcpy.Describe(fcName1)

    #Check the spatial reference for arbordaze2009tents
    fcName2 = "arbordaze2009tents"
    fcName2Properties = arcpy.Describe(fcName2)

    #Check the spatial reference for complan
    fcName3 = "complan"
    fcName3Properties = arcpy.Describe(fcName3)

    #Print the results
    print fcName1Properties.spatialReference.name
    print fcName2Properties.spatialReference.name
    print fcName3Properties.spatialReference.name

except:
    print "I couldn't find that feature class!"
```

This example is simplistic error handling, and you will learn more about error handling later.

Exercise 2-1

Write the pseudo code and script to discover the shape type in the specified feature class name. (**Hint:** Search ArcGIS for Desktop Help for the shapeType property of the Describe() function, which will explain the syntax for using this function. Check the describe object properties for a property to get the file's base name and the feature class properties for a property to get the file's shape type.)

Tutorial 2-1 review

The format for this Python script is the basis for every Python script you write, with the documentation first, the import arcpy statement (and any other module you need) coming next, and then the code to perform the operation. You can make a template file of this script format for future use. This template will not only simplify your code writing in the future, but also standardize your code. The inclusion of comments is important to help document your pseudo code. These comments also are helpful if you or another programmer must alter this script at some point in the future. This alteration may be made weeks or years later, and relearning the process this code is performing may be difficult without these comments.

The use of the Describe() function is also an important process to learn. Many of the properties of your data can be accessed by describing them to an object. ArcGIS for Desktop Help is helpful in finding all these properties. It is important to note that there may be more than one describe group for any one item, so searching Help is a good way to find all the groups. For instance, you could describe a feature class to an object and return characteristics as shown under properties, including Describe Object Properties, Dataset Properties, FeatureClass Properties, and File Properties.

Study questions

1. Using ArcGIS for Desktop Help, how many ways can you find to access the describe properties of a table?
2. What is the ArcPy module, and why is it used for geoprocessing scripts? Give examples of the modules, classes, and functions contained in the ArcPy module.
3. How would you find the available properties in the arcpy.environment class?

Tutorial 2-2 Scripting geoprocessing tasks

Almost any geoprocessing task can be automated with Python. This automation can help increase productivity for full-time GIS users or allow complex tasks to be shared with casual GIS users.

Learning objectives

- Programming geoprocessing tasks
- Understanding tool syntax

Preparation

Research the following topics in ArcGIS for Desktop Help:

- "Using the Results window"
- "Union (Analysis)"
- "Select (Analysis)"

Introduction

The environment settings and describe objects you worked with in tutorial 2-1 demonstrate the characteristics of feature classes. Now you are ready to start using these feature classes in geoprocessing tasks.

All the geoprocessing tools in ArcGIS are available through ArcPy. Each tool description provides both simple and complex Python code examples so that you can get an idea of how to use the tool in your scripts. Simply copy the examples into your own scripts, and modify them to use different workspaces and datasets.

Run the tools in ArcMap to test your processes. Then copy a code snippet from the process results to use in your scripts. For the more adventurous, drop the tool into a model, configure it, and export the model to a Python script. If done tool by tool, you could systematically extract almost all the code needed for a new script.

Scenario

You are creating a new wall map for the city planner that she intends to hang in her office. The map should show the parcels color-coded by the zoning classification. The datasets you have are parcels that have neither the zoning codes nor the zoning districts data that overlaps the streets. The objective is to create a dataset to use in this map that will color-code each parcel by its zoning classification without including the streets.

To create this dataset, union the parcels and the zoning districts to create a new feature class, and then remove the segments that are in the street right-of-way.

Data

The map document for this tutorial has the property lines for reference, the parcel data, and the zoning districts data, which is already classified by zoning code. The workspace for this data is as follows:

C:\EsriPress\GISTPython\Data\City of Oleander.gdb

(Modify it if necessary to match the location where you installed the student data.)

SCRIPTING TECHNIQUES

In this tutorial, you will start bringing geoprocessing tools into your script. There is no real trick to it, but it is good practice not only to look up the tool in ArcGIS for Desktop Help and understand its usage, but also to copy and paste the syntax into your script as a comment. This comment will provide an easily accessible reference for future use.

Also used in the script is a technique to run your tools across several lines in the script. This is accomplished by adding a space and a backslash (\) after any of the parameters and then pressing Enter. The code is shown on multiple lines but will run as though it is on one. It is good practice to start a new line of code after one of the commas separating the parameters rather than in the middle of a path name or query statement.

Script geoprocessing tasks

1. **Write the pseudo code for this script. A completed version of the pseudo code is shown at the end of this tutorial. Write your pseudo code, and then compare it with the pseudo code provided. If your pseudo code differs and seems viable, work the tutorial as written, and then go back and try it again with your unique pseudo code.**

2. **Open the map document Tutorial 2-2. Note that the zoning categories shown overlap the streets—that is the part you will remove.**

3. **Start a new Python script in your IDE, and name it ZoneUnion.py. Add all the standard lines of code that you will need in your script, as shown:**

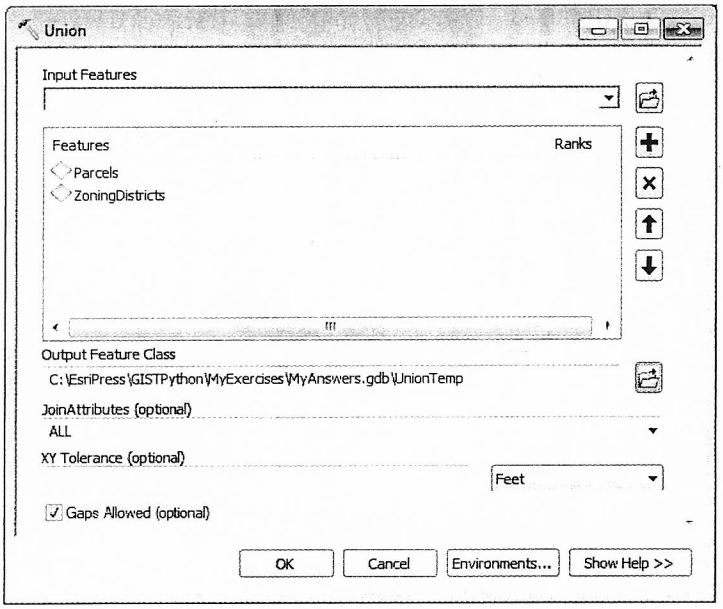
```
#---------------------------------------------------------------------------
# Name:        Tutorial 2-2
# Purpose:     Union two feature classes and delete streets
#
# Author:      David W. Allen, GISP
#
# Created:     10/25/2013
# Copyright:   (c) David 2013
#
#---------------------------------------------------------------------------

try:
    # Import the modules
    import arcpy
    from arcpy import env

    # Set up the environment

    # ENTER CODE

    # Determine results

except:
    print "Message"
```

This template of code can be the basis for almost every script you write.

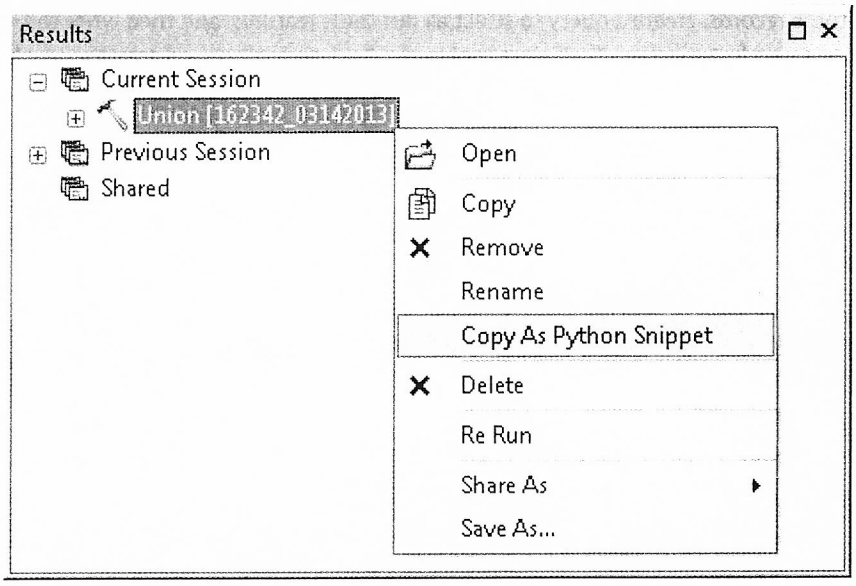
4. **Add the code to set the workspace environment setting to the location of the parcels and zoning data. For help, refer to tutorial 2-1.**

Next, union the parcels and the zoning districts layers. There are several ways to get a code snippet of this tool, so for this tutorial, use the geoprocessing Results window.

5. **In ArcMap, open the Search window, and find the Union tool. Run this tool with the parameters shown:**



6. **When the tool is finished running, open the Geoprocessing > Results window from the main ArcMap toolbar. Expand the Current Session line, right-click Union, and click Copy As Python Snippet, as shown. Close the Results window.**



7. **In your IDE, start a new indented line after the ENTER CODE comment line, right-click, and click Paste. In the example shown, a backslash was added to wrap the code onto a second line.**

```
    # ENTER CODE
    arcpy.Union_analysis("Parcels #;ZoningDistricts #",\
    "C:/EsriPress/GISTPython/MyExercises/MyAnswers.gdb/UnionTemp","ALL","#","GAPS")
```

You can see that the code for the Union tool starts with arcpy to access the ArcPy module, and then includes the tool name. Next are an underscore and the word *analysis*, which calls out the alias of the toolbox that contains this tool. A set of parentheses contains all the parameters for running this tool. The snippet will contain the raw path, so add a lowercase *r* at the beginning of the path to signal Python to interpret this string as a path and not as a string with special escape characters.

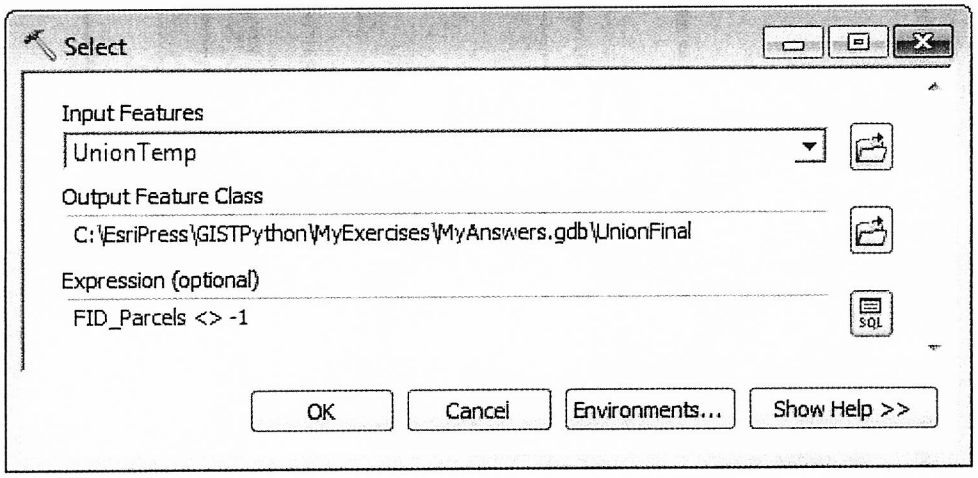
Notice that because you used a different file location for the output than the environment you set earlier, you will need to include this entire string when this file is used in other commands.

Also, be careful using this process of getting code snippets when you are writing a stand-alone script. The code snippet will use the layer alias from the table of contents, but when you run the script, it will look for the actual layer name. Any layer with an alias will need to be replaced in the code with the actual layer name.

8. **In ArcMap, open the attribute table of the UnionTemp layer. Right-click the FID_Parcels field and click Sort Ascending. Notice the records with a value of -1. Close the attribute table.**

The records with a value of -1 are the parts of the zoning data that fall in the street right-of-way. To remove these records, create a query to select all but these features, and then write the selected features to a new feature class. The Select tool in the Analysis toolbox will create a new feature class that contains only those features that meet a selection query.

9. **Use the Search window to find the Select tool. Run the tool, set the tool to output only the records from UnionTemp where FID_Parcels are <> -1, and store the records in a new feature class named UnionFinal. When your parameters match those shown in the graphic, click OK.**



10. **Using the technique shown in step 6, copy the Python snippet for the Select tool from the geoprocessing Results window.**

11. **In your IDE, add a new indented line below the Union command, and paste the code snippet for the Selection tool. Add the full path for the UnionTemp layer in the command and the *r* at the beginning of the paths. Some additional comment lines are added, which you can delete.**

```
arcpy.Select_analysis(r"C:/EsriPress/GISTPython/MyExercises/MyAnswers.gdb/UnionTemp",\
"C:/EsriPress/GISTPython/MyExercises/MyAnswers.gdb/UnionFinal","FID_Parcels <> -1")
```

Note: Versions of ArcGIS prior to 10.2.1 may not format the query string in this command correctly. To accommodate this deviation, you must rework the query to use single or double quotation marks as shown:

```
arcpy.Select_analysis(r"C:/EsriPress/GISTPython/MyExercises/MyAnswers.gdb/UnionTemp",\
"C:/EsriPress/GISTPython/MyExercises/MyAnswers.gdb/UnionFinal",'"FID_Parcels" <> -1')
```

The output will be a feature class that shows zoning for each parcel but not in the street right-of-way.

12. **Add a print statement after the "Determine results" comment, and change the error message at the end to something more appropriate for error handling, as shown:**

```
try:
    # Import the modules
    import arcpy
    from arcpy import env

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

    # ENTER CODE
    arcpy.Union_analysis("Parcels #;ZoningDistricts #",\
    "C:/EsriPress/GISTPython/MyExercises/MyAnswers.gdb/UnionTemp","ALL","#","GAPS")

    arcpy.Select_analysis(r"C:/EsriPress/GISTPython/MyExercises/MyAnswers.gdb/UnionTemp",\
    "C:/EsriPress/GISTPython/MyExercises/MyAnswers.gdb/UnionFinal",'"FID_Parcels" <> -1')

    # Determine results
    print "New feature class has been created"

except:
    print "Process did not complete"
```

Next, delete the file UnionTemp from the geodatabase.

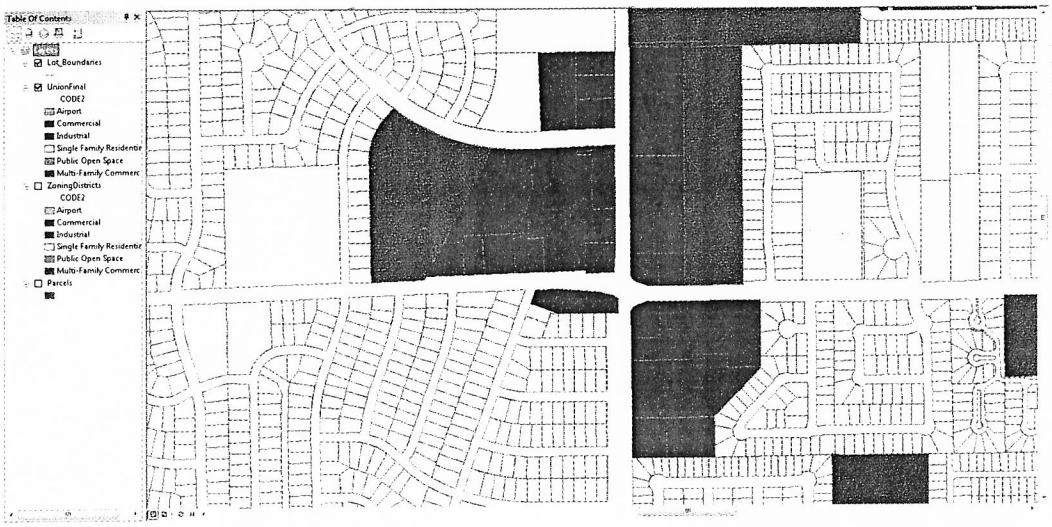
Your turn

Use the Search window to find a tool that will delete the feature class UnionTemp from the MyAnswers geodatabase. Configure the tool correctly, and add it to your script. Pay attention to the path names, and use the lowercase r if necessary to identify the path.

13. **In ArcMap, use the Catalog window to delete the feature classes UnionTemp and UnionFinal that you created in testing. In your IDE, save your code, and run it, as shown:**

```
#-------------------------------------------------------------------------------
# Name:        ZoneUnion.py
# Purpose:     Union two feature classes and delete streets
#
# Author:      David W. Allen, GISP
#
# Created:     10/25/2013
# Copyright:   (c) David 2013
#
#-------------------------------------------------------------------------------

try:
    # Import the modules
    import arcpy
    from arcpy import env

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

    # ENTER CODE
    arcpy.Union_analysis("Parcels #;ZoningDistricts #",\
    "C:/EsriPress/GISTPython/MyExercises/MyAnswers.gdb/UnionTemp","ALL","#","GAPS")

    arcpy.Select_analysis(r"C:/EsriPress/GISTPython/MyExercises/MyAnswers.gdb/UnionTemp",\
    "C:/EsriPress/GISTPython/MyExercises/MyAnswers.gdb/UnionFinal",'"FID_Parcels" <> -1')

    arcpy.Delete_management("C:/EsriPress/GISTPython/MyExercises/MyAnswers.gdb/UnionTemp","FeatureClass")

    # Determine results
    print "New feature class has been created"

except:
    print "Process did not complete"
```

When your script is finished running, a new feature class will be created that meets the city planner's requirements. Add the script to your map document, and import the same symbol schema from the ZoningDistricts layer.



Here's the pseudo code for this project:

```
# Import the ArcPy module and env class
# Set up the working environment
# Perform the Union of the Parcel and Zoning District layers
# The parcel polygons that do not overlap the zoning polygons
# will be the street right-of-way and have an ID of -1
# Select out the parcels with an ID of -1
# Output a new layer with the results
# Symbolize to match the existing zoning layer
```

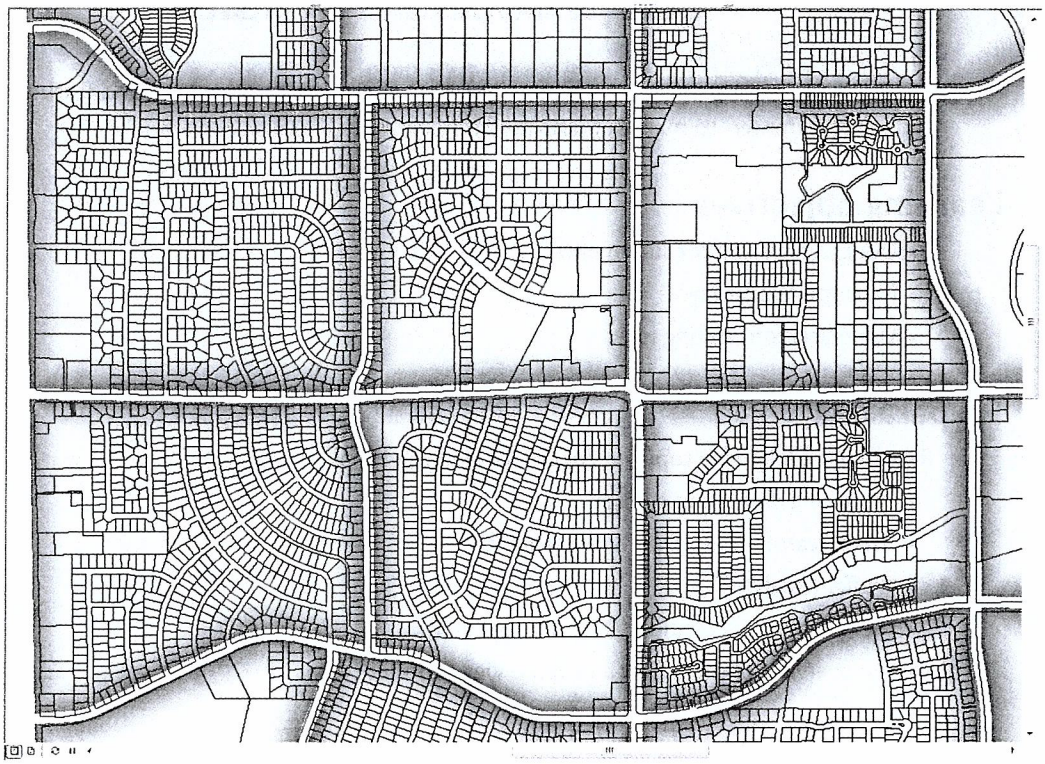
Exercise 2-2

The fire chief needs an updated map of the Fire Department's response zones, called *boxes*. The data layer FireBoxMap contains these zones. He would like the zones to have a gap between them so that they stand out on the printed maps. For aesthetics, you could symbolize the zones with a graduated color at the edges. The problem is, how is the feature for this type of display created?

Here is a description of the process from which you can write your pseudo code, research the tools necessary, and eventually write the script:

Convert the FireBoxMap polygons to a linear feature class, buffer them 50 feet, union the result with the FireBoxMap layer, and remove the features that represent the gap. Have your script delete any temporary files made in the process.

If done correctly, it should look like this:



Tutorial 2-2 review

This tutorial uses some geoprocessing tools that you are probably familiar with from projects you may have done in ArcMap. When running the tools manually, you filled in the parameters in a pop-up dialog box and started the overlay process. The geoprocessing Results window provided a code snippet that you can use in your own scripts, which is convenient because all the tool's required and optional parameters are already set. Having this reference is also a good way to study the code and see how various paths and values are handled.

Study questions

1. This process of color-coding each parcel by zoning classification used the Union tool. Could you have used a different overlay tool? If so, what changes would need to be made in the code? Create a code snippet in the geoprocessing results for your code.
2. Could you have found all the polygons in the output of the Union tool that represented the right-of-way? Add code to make that layer, and store it in a new feature class.
3. The input layers for the Union tool must be polygons. Can you write the code to make sure that the layers are of the type polygon before the Union tool runs? (**Hint:** look at the describe properties.)

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		W
1	5217	MIDWAY PK	BLK 18 LOT 10	0.996592	0	1	F1	25840 18 10	W	H
2	6652	MIDWAY SQUARE ADDITION	BLK A LOT 18	0.163853	1	1	A5	25975 A 18		EF
3	6658	MIDWAY SQUARE ADDITION	BLK A LOT 17	0.154063	1	1	A5	25975 A 17		EF
4	7196	WOODCREEK	BLK A LOT 43	0.102241	1	1	A5	47485 A 43		R
5	7204	HARWOOD COURTS	BLK E LOT 4	0.103758	1	1	A5	17402 E 4		BF
6	7481	BEAR CREEK BEND ADDITION	BLK A LOT 32	0.207402	1	1	A1	1899 A 32		W
7	7482	BEAR CREEK BEND ADDITION	BLK A LOT 33	0.142065	1	1	A1	1899 A 33		W
8	7483	BEAR CREEK BEND ADDITION	BLK A LOT 34	0.114024	1	1	A1	1899 A 34		W
9	7487	BEAR CREEK BEND ADDITION	BLK A LOT 35	0.13864	1	1	A1	1898 A 35		W
10	7496	HARWOOD CROSSING	BLK 1 LOT 3	4.31566	0	1	F1	17403 1 3	N	M
11	6513	MIDWAY SQUARE ADDITION	BLK B LOT 3	0.160455	1	1	A5	25975 B 3		EF
12	6520	MIDWAY SQUARE ADDITION	BLK B LOT 4	0.162595	1	1	A5	25975 B 4		EF
13	6741	ARBOR GLEN ADDITION	BLK A LOT 72	0.066816	1	1	A4	796C A 72		AI
14	6743	ARBOR GLEN ADDITION	BLK A LOT 81	0.092001	1	1	A4	796C A 81		AI
15	6437	MIDWAY SQUARE ADDITION	BLK A LOT 25	0.152216	1	1	A5	25975 A 25		IN
16	6631	MIDWAY SQUARE ADDITION	BLK A LOT 19	0.304003	1	1	A5	25975 A 19		EF
17	6903	WOODCREEK	BLK E LOT 32	0.20849	4	1	B4	47485 E 32	E	DE
18	6376	MIDWAY SQUARE ADDITION	BLK C LOT 3	0.161747	1	1	A5	25975 C 3		BF
19	6407	MIDWAY SQUARE ADDITION	BLK A LOT 24	0.16022	1	1	A5	25975 A 24		IN
20	6415	MIDWAY SQUARE ADDITION	BLK A LOT 23	0.167889	1	1	A5	25975 A 23		BF
21	6439	MIDWAY SQUARE ADDITION	BLK B LOT 1	0.18062	1	1	A5	25975 B 1		BF
22	6448	MIDWAY SQUARE ADDITION	BLK A LOT 22	0.151085	1	1	A5	25975 A 22		BF

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
```

Exercise 2-3

The city manager has been asked by the city council to determine whether there are any areas of town that do not have adequate nighttime illumination from street lights. He in turn has passed this task on to you. Your results will be used to determine where street lights need to be added or whether any existing lights should be moved in the long term. You must write a script to do this job because the locations will be changing over the next two years, and you will be running this analysis frequently.

The data in the map document Exercise 2-3 includes the current street light locations. There are four different types of lights, and each type requires a different buffer distance. The codes are stored in a field named Type and are listed here with their appropriate buffer distances:

MV (mercury vapor) = 125 ft

MVH (mercury vapor—high pressure) = 160 ft

SV (sodium vapor) = 100 ft

SVH (sodium vapor—high pressure) = 200 ft

The script you write will need to consider each feature and generate a buffer according to the light type. The final results can be overlaid on the city layer to reveal places that need more lighting. Creating a new feature dataset to hold the output files would be helpful.

Write the pseudo code and the script necessary to perform this task. (**Hint:** Try adding a field to the feature class to store the buffer distance, and then go through each feature with a cursor and determine which buffer distance should be used. When all the distances are set, run the buffer tool using the new attribute as the buffer distance.)

Tutorial 2-3 review

When used in a script, the tool parameters must be set in the code. These parameters can be a hard-coded value or even a variable. The required and optional parameters can be found by looking up the tool in ArcGIS for Desktop Help. The Help files also include sample scripts showing how the tools can be used in your own scripts. Note that some of the parameters may be optional and can either be included, left out completely, or skipped by providing the value of "" as a placeholder so that other optional parameters can be accessed further along in the tool's usage syntax.

This tutorial also uses cursors to access the data feature by feature. Cursors can be used to access the data or to change or add data. The syntax is to create the cursor object, which holds all the values of all the features in one place. The for statement can then be used to step through the features or rows one at a time for processing. Closing the cursor at the end of its use is important so that the features are not locked to further access.