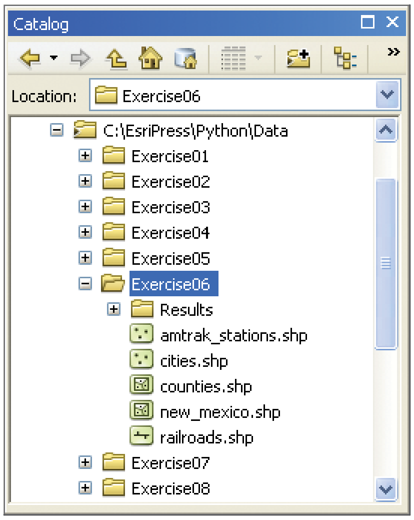
Exercise 6
Exploring spatial data

Check for the existence of data

Before starting to work with the exercise data, you will preview the data.

1 **Start ArcMap with a new empty map document. On the Standard toolbar, click the Catalog button to open the Catalog window.**

2 **Browse to the C:\EsriPress\Python\Data\Exercise06 folder.** →



Notice that there are five shapefiles in this folder, including point, polyline, and polygon shapefiles.

3 **Start PythonWin. Create a new Python script and save as** shape_exists.py **to the C:\EsriPress\Python\Data\Exercise06\Results folder.**

4 **Enter the following lines of code:**

```
import arcpy
from arcpy import env
env.workspace = "C:/EsriPress/Python/Data/Exercise06"
shape_exists = arcpy.Exists("cities.shp")
print shape_exists
```

5 **Save and run the script.** Running the script returns a value of `True`.

6 **Modify the script by replacing** `"cities.shp"` **with** `"CITIES.SHP"`**.**

7 Save and run the script. Running the script returns the value of True. Python, for the most part, is case sensitive and it applies to strings as well. One of the exceptions is path and file names, so "cities.shp" is the same as "CITIES.SHP" and "C:/EsriPress/PYTHON/DATA/EXERCISE06" is the same as "c:/EsriPress/python/data/exercise06".

Checking for the existence of data is a function that is commonly used in stand-alone scripts.

8 Modify the script as follows:

```
import arcpy
from arcpy import env
env.workspace = "C:/EsriPress/Python/Data/Exercise06"
if arcpy.Exists("cities.shp"):
    arcpy.CopyFeatures_management("cities.shp", "results/cities_copy.➜
➜ shp")
```

9 Save and run the script. The script determines whether the particular input feature class exists and runs the geoprocessing operation accordingly.

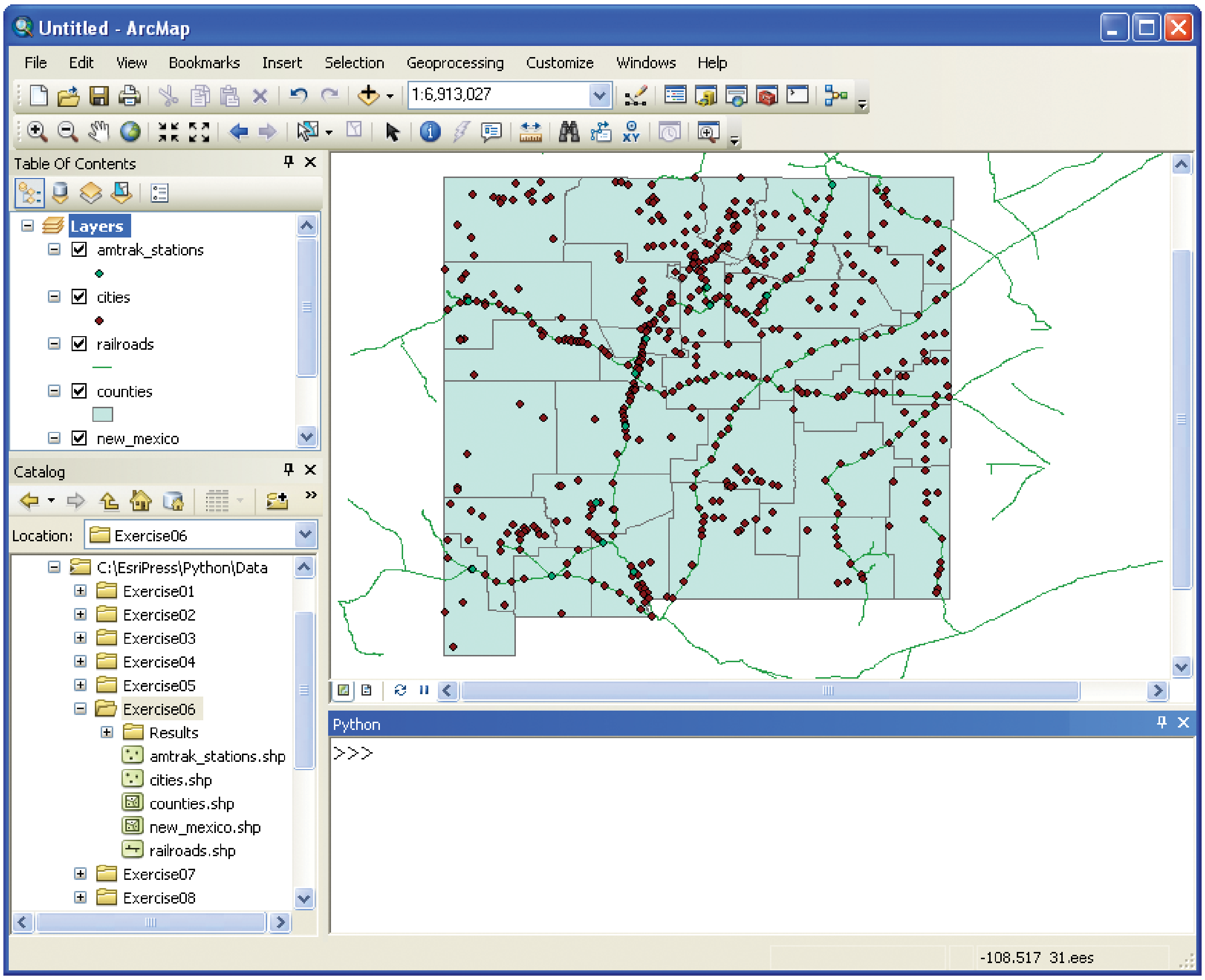
10 Close PythonWin.

Describe the data

The Describe function can be used to determine properties of datasets and other inputs into geoprocessing tools.

1 In ArcMap, open the Catalog and Python windows.

2 Drag the five feature classes from the Exercise06 folder into the map document.



3 Run the following code:

```
    >>> myshape = arcpy.Describe("C:/EsriPress/Python/Data/Exercise06/ ➔
➔ cities.shp")
```

You can now access the properties of the object.

4 Run the following code:

```
    >>> myshape.dataType
```

Running the code returns `u'Shapefile'` as the data type of the object. You can also work with the layers in the current map document.

5 Run the following code:

```
>>> mylayer = arcpy.Describe("cities")
>>> mylayer.dataType
```

Running the code returns `u'FeatureLayer'`. This is the same shapefile but now accessed as a layer from the current map. You will examine a few more properties in steps 6–8.

6 Run the following code:

```
>>> mylayer.datasetType
```

The result is `u'FeatureClass'`.

```
>>> mylayer.catalogPath
```

The result is as follows:

```
u'C:\\EsriPress\\Python\\Data\\Exercise06\\cities.shp'
```

```
>>> mylayer.basename
```

The result is `u'cities'`.

```
>>> mylayer.file
```

The result is `u'cities.shp'`.

```
>>> mylayer.isVersioned
```

The result is `False`.

```
>>> mylayer.shapeType
```

The result is `u'Point'`.

Most properties consist of strings or Boolean values, and simply accessing the property prints its value. Some properties, however, consist of objects, and these can have many properties, which need to be accessed separately.

For example, accessing the `spatialReference` property of the feature class returns a `SpatialReference` object.

7 Run the following code:

```
>>> mylayer.spatialReference
```

The result is as follows:

```
<geoprocessing spatial reference object object at 0x101CF3E0>
```

Notice that the code returns a reference to an object. The reference value will vary with every session, and in itself is not very useful. However, the object has many properties, which can each be accessed individually.

8 Run the following code:

```
>>> mylayer.spatialReference.name
```

The result is `u'GCS_North_American_1983'`.

Note: *A regular string preceded by the letter* u *is called a Unicode string. Unicode strings work just like regular strings but are more robust when working with different international sets of characters.*

```
>>> mylayer.spatialReference.type
```

The result is `u'Geographic'`.

```
>>> mylayer.spatialReference.domain
```

The result is `u'-400 -400 400 400'`.

9 Close ArcMap. There is no need to save your map document.

List the data

Describing data typically works on a single element, such as a feature class. List functions can be used to work with many types of elements, including feature classes, rasters, tables, and fields.

1 Start PythonWin. Create a new Python script and save as list.py **to the Results folder for exercise 6.**

2 Enter the following lines of code:

```
import arcpy
from arcpy import env
env.workspace = "C:/EsriPress/Python/Data/Exercise06"
fclist = arcpy.ListFeatureClasses()
print fclist
```

***Note:** The preceding lines of code could just as easily be run using the Python window in ArcMap. In general, switching between the Python window and PythonWin is a matter of preference. However, writing scripts in an editor like PythonWin is typically preferable as your scripts get longer and so you can make changes to your scripts in the future.*

3 Save and run the script.

The list of feature classes is printed to the Interactive Window, as follows:

```
>>> [u'amtrak_stations.shp', u'cities.shp', u'counties.shp', u'new_ ➜
➜ mexico.shp', u'railroads.shp']
```

Once a list of elements is obtained, a `for` loop can be used to iterate over the elements of the list and carry out a specific task. For example, the `Describe` function can be used to access properties of each of the feature classes in a workspace.

4 Modify the script as follows:

```
import arcpy
from arcpy import env
env.workspace = "C:/EsriPress/Python/Data/Exercise06"
fclist = arcpy.ListFeatureClasses()
for fc in fclist:
    fcdescribe = arcpy.Describe(fc)
    print "Name: " + fcdescribe.name
    print "Data type: " + fcdescribe.dataType
```

5 Save and run the script.

Running the script prints the name and data type of each feature class to the Interactive Window, as follows:

```
Name: amtrak_stations.shp
Data type: ShapeFile
Name: cities.shp
Data type: ShapeFile
Name: counties.shp
Data type: ShapeFile
Name: new_mexico.shp
Data type: ShapeFile
Name: railroads.shp
Data type: ShapeFile
```

The same approach can be used to work with geoprocessing tools.

6 **Save your list.py script as** listcopy.py **to the Results folder for exercise 6.**

7 **Modify the script as follows:**

```
    import arcpy
    from arcpy import env
    env.workspace = "C:/EsriPress/Python/Data/Exercise06"
    fclist = arcpy.ListFeatureClasses()
    for fc in fclist:
        arcpy.CopyFeatures_management(fc, "C:/EsriPress/Python/Data/➜
➜   Exercise06/Results/" + fc)
```

8 **Save and run the script.**

9 **In ArcMap, examine the result in the Catalog window. Confirm that the shapefiles have been copied to the Results folder.** Running the script copies the shapefiles without modifying their names. In some cases, the names need to be modified—for example, when copying shapefiles to a geodatabase. The script you will write next creates a new empty file geodatabase called NM.gdb and copies the shapefiles from the workspace to this new geodatabase. However, the name of a shapefile, by default, includes the file extension ".shp," which needs to be removed. The script therefore uses the basename property of the feature class rather than the default name for the name of the shapefile.

>>> TIP

To see the result in the Catalog window, right-click a folder (in this case, the Results folder for exercise 6) and click Refresh. This makes any newly added datasets visible.

10 Modify the script as follows:

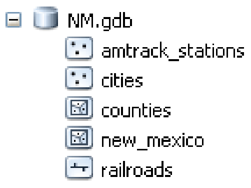
```
    import arcpy
    from arcpy import env
    env.overwriteOutput = True
    env.workspace = "C:/EsriPress/Python/Data/Exercise06"
    arcpy.CreateFileGDB_management("C:/EsriPress/Python/Data/Exercise06/ ➜
➜   Results", "NM.gdb")
    fclist = arcpy.ListFeatureClasses()
    for fc in fclist:
        fcdesc = arcpy.Describe(fc)
            arcpy.CopyFeatures_management(fc, "C:/EsriPress/Python/Data/ ➜
➜   Exercise06/Results/NM.gdb/" + fcdesc.basename)
```

>>> TIP

Some scripts use a statement such as `rstrip(".shp")` to remove the file extension, but this can inadvertently remove additional letters from the name. Using the `basename` property is therefore recommended.

11 Save and run the script.

12 Examine the result in the Catalog window. It should look like the example in the figure. ➜


NM.gdb
amtrack_stations
cities
counties
new_mexico
railroads

13 Close ArcMap. There is no need to save your map document.

List functions exist for several different types of elements, including fields. Next, you will create a script for listing the fields of the cities shapefile.

14 In PythonWin, create a new Python script. Save as listfields.py **to the Results folder for exercise 6.**

15 Enter the following lines of code:

```
    import arcpy
    from arcpy import env
    env.overwriteOutput = True
    env.workspace = "C:/EsriPress/Python/Data/Exercise06"
    fieldlist = arcpy.ListFields("cities.shp")
    for field in fieldlist:
        print field.name + " " + field.type
```

16 Save and run the script. The `ListFields` function returns a list of field objects. The `name` property of these objects is used to print the names of the fields.

Running the script prints a list of the names of the fields in the feature followed by their type. The result is as follows:

```
FID OID
Shape Geometry
CITIESX020 Double
FEATURE String
NAME String
POP_RANGE String
POP_2000 Integer
FIPS55 String
COUNTY String
FIPS String
STATE String
STATE_FIPS String
DISPLAY SmallInteger
```

17 Close PythonWin. There is no need to save the results in the Interactive Window.

As you have been working through the exercises in this book, you have probably encountered some unexpected errors. A simple typo can cause a script not to run properly. Even when all typos are fixed, you may continue to run into errors. One of the most common error messages is as follows:

```
ExecuteError: ERROR 000258: Output C:\<folder>\<file> already exists.
```

What happens quite often is that you run a script, and then modify it and try running it again. The script then tries to overwrite existing files that resulted from the earlier execution of the script, and the script fails. To overcome these types of errors, you can add the following line to your script:

```
env.overwriteOutput = True
```

This line of code makes it possible for the script to overwrite existing files. Even with this statement, however, error messages of this type may continue. If you are running a stand-alone script from PythonWin and are also using ArcMap or ArcCatalog to examine the results, ArcGIS for Desktop may have placed a shared lock on the data, preventing it from being overwritten. This is a common error when working with geodatabases in particular.

>>> TIP

To overcome error messages related to a shared lock on the data, close all ArcGIS for Desktop applications and run the script again. When the script is finished running, you can open ArcMap or ArcCatalog to examine the results.

Manipulate lists

Lists are widely used in batch geoprocessing. Lists can be manipulated in a number of different ways.

1 Start ArcMap and open the Python window.

2 Run the following code:

```
>>> arcpy.env.workspace = "C:/EsriPress/Python/Data/Exercise06"
>>> fclist = arcpy.ListFeatureClasses()
>>> print fclist
```

Running the code prints a list of feature classes in the list, as follows:

```
[u'amtrak_stations.shp', u'cities.shp', u'counties.shp', u'new_mexico. ➜
➜ shp', u'railroads.shp']
```

Any list in Python can be manipulated using the built-in Python functions and methods. Python lists are indexed starting with the number zero (0). This makes it possible to obtain specific elements in the list or to use slicing functions to create smaller lists that contain just the desired elements. You'll use indexing next.

3 Run the following code:

```
>>> fclist[0]
```

The result is `u'amtrak_stations.shp'`.

```
>>> fclist[3]
```

The result is `u'new_mexico.shp'`.

```
>>> fclist[-1]
```

The result is `u'railroads.shp'`.

```
>>> fclist[1:3]
```

The result is `[u'cities.shp', u'counties.shp']`.

```
>>> fclist[2:]
```

The result is as follows:

```
[u'counties.shp', u'new_mexico.shp', u'railroads.shp']
```

You can also create a list by typing the elements of the list, which you'll do next.

4 Run the following code:

```
>>> cities = ["Alameda", "Brazos", "Chimayo", "Dulce"]
```

The number of features can be determined using the `len` function, which you'll use next.

5 Run the following code:

```
>>> len(cities)
```

The result is `4`.

The `del` statement removes one or more elements from the list. Because this code does not automatically return the list, a print statement is used to view the current list.

6 Run the following code:

```
>>> del cities[2]
>>> print cities
```

The result is `['Alameda', 'Brazos', 'Dulce']`.

The `sort` method can be used to sort the elements in a list, and it can also be reversed. You'll try both methods next.

7 Run the following code:

```
>>> cities.sort(reverse = True)
>>> print cities
```

The result is `['Dulce', 'Brazos', 'Alameda']`.

```
>>> cities.sort()
>>> print cities
```

The result is `['Alameda', 'Brazos', 'Dulce']`.

Determining list membership is accomplished using the `in` operator, which you'll use next.

8 Run the following code:

```
>>> "Zuni" in cities
```

The result is `False`.

The `append` method can be used to add a new element to the end of the list, and the `insert` method makes it possible to add a new element at a given location, which you'll try next.

9 Run the following code:

```
>>> cities.append("Zuni")
>>> print cities
```

The result is `['Alameda', 'Brazos', 'Dulce', 'Zuni']`.

```
>>> cities.insert(0,"Espanola")
>>> print cities
```

The result is as follows:

```
['Espanola', 'Alameda', 'Brazos', 'Dulce', 'Zuni']
```

Work with dictionaries

Dictionaries are like lookup tables: they consist of pairs of keys and their corresponding values. Keys are unique within a dictionary although values may not be. Keys must be of an immutable data type, such as strings, numbers, or tuples, although values can be of any type.

1 Run the following code:

```
>>> countylookup = {"Alameda": "Bernalillo County", "Brazos": "Rio ➔
➔ Arriba County", "Chimayo": "Santa Fe County"}
```

This dictionary consists of pairs of cities (the keys) and their corresponding counties. Cities as the keys are unique, whereas the county values are not—that is, unique cities may be located in the same county. Notice the syntax, as follows:

- The entire dictionary is contained by curly brackets ({ }).
- Keys are separated from their values by a colon (:).
- Pairs of keys and values are separated from each other by a comma (,).
- Strings are enclosed in double quotation marks (" ").

Once created, the dictionary can be used as a lookup table, which you'll try next.

2 Run the following code:

```
>>> countylookup["Brazos"]
```

The result is `'Rio Arriba County'`.

Notice that the syntax for dictionaries is similar to working with elements in a list. The name of the dictionary is followed by square brackets ([ ]), but instead of an index number inside the brackets, one of the keys is used.

The dictionary is designed to work one way only, which you'll see next. You can only search a dictionary by its keys to get the corresponding values. You cannot do the reverse and search for a value to get a key.

3 Run the following code:

```
>>> countylookup["Santa Fe County"]
```

The code results in an error, as follows:

```
Runtime error
Traceback (most recent call last):
  File "<string>", line 1, in <module>
KeyError: 'Santa Fe County'
```

In this case, `"Santa Fe County"` is not one of the keys, and running the code thus returns an error.

Several additional operations can be performed on dictionaries. For example, the number of pairs can be obtained using the `len` function, as follows:

```
>>> len(countylookup)
```

The result is `3`.

The pairs in the dictionaries can also be split using the `key` and `value` methods. The `key` method returns the keys of the dictionaries as a list, and the `value` method returns the corresponding values as a list, which you'll see next.

```
>>> countylookup.keys()
```

The result is `['Chimayo', 'Alameda', 'Brazos']`.

```
>>> countylookup.values()
```

The result is as follows:

```
['Santa Fe County', 'Bernalillo County', 'Rio Arriba County']
```

Challenge exercises

Challenge 1

Write a script that reads all the feature classes in a workspace and prints the name of each feature class and the geometry type of that feature class in the following format:

```
streams is a point feature class
```

Challenge 2

Write a script that reads all the feature classes in a personal or file geodatabase and copies only the polygon feature classes to a new file geodatabase. You can assume there are no feature datasets in the existing data, and the feature classes can keep the same name.