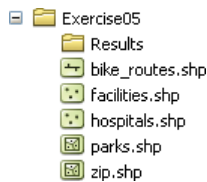# Exercise 5
Geoprocessing using Python

Use tools

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

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 exercise 5 folder.**

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

3 **Drag the parks.shp and zip.shp files to the ArcMap Table Of Contents window.**

4   Dock the Catalog window at the bottom of the Table Of Contents window.
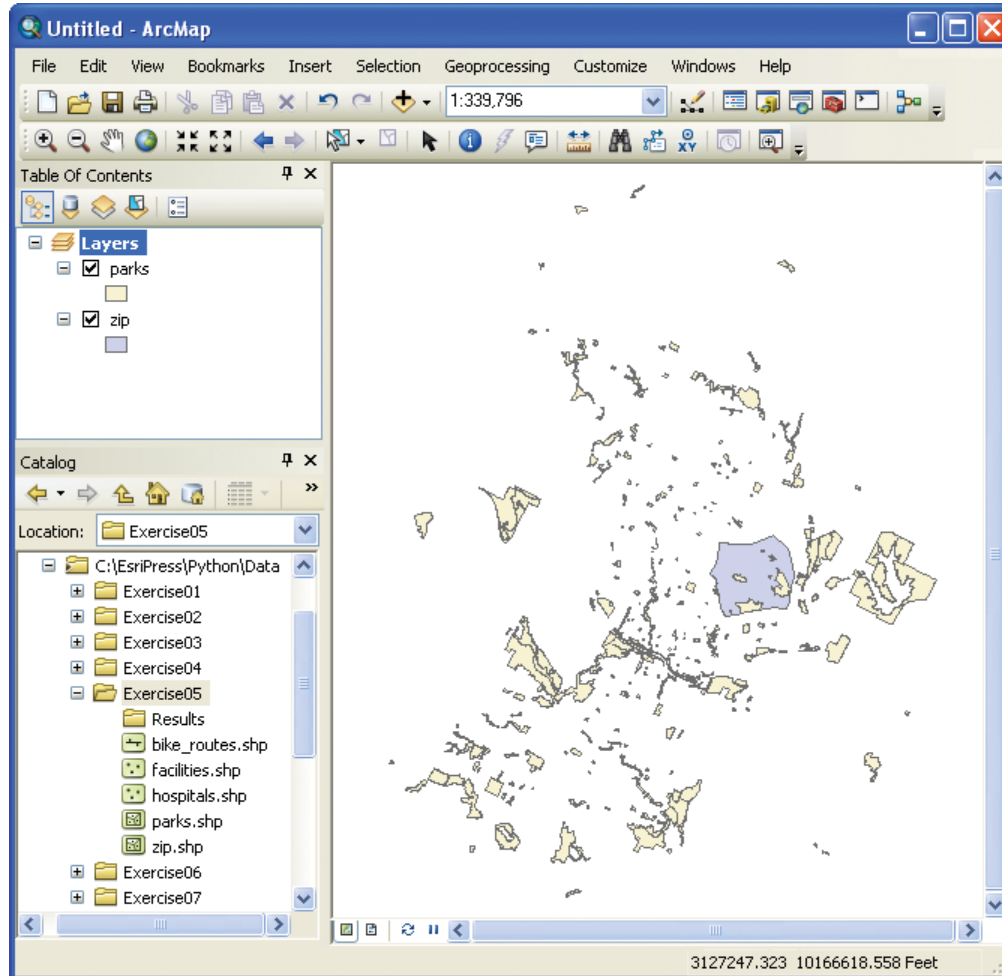
5   On the Standard toolbar, click the Python button to open the Python window.

6   Dock the Python window below the map display area.
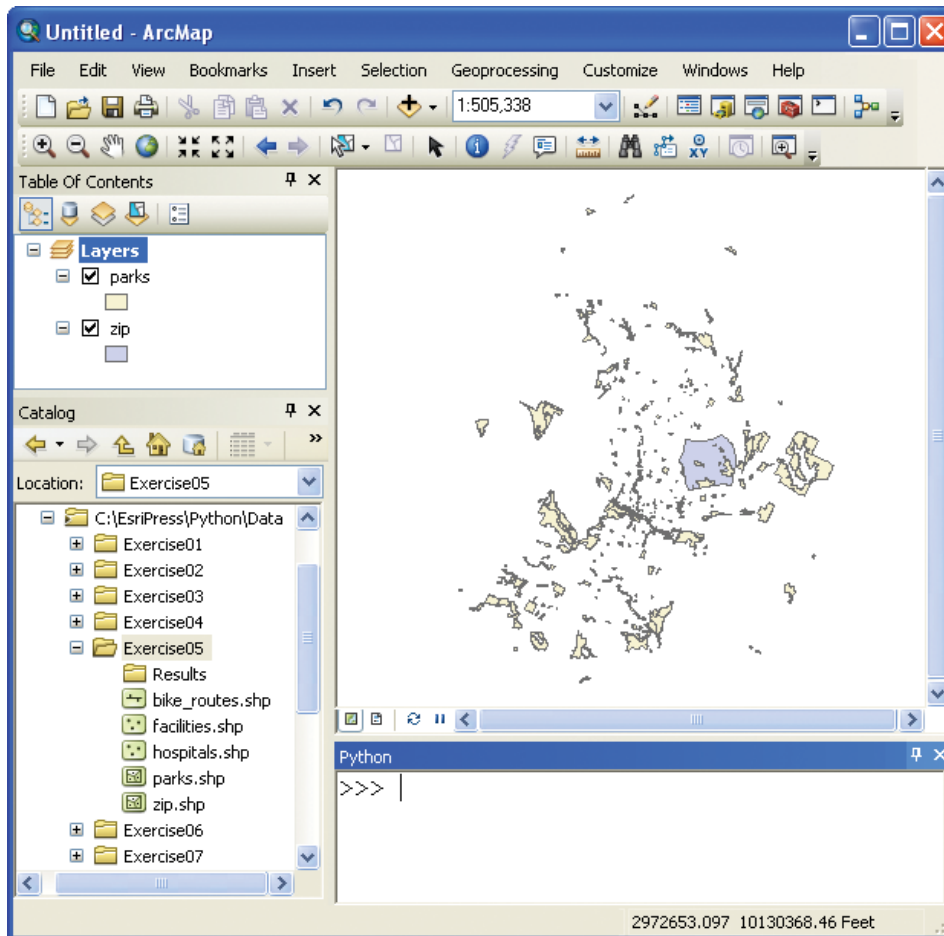
Note: Docking the Catalog and Python windows is not required, but it can help to organize your available desktop space. Typically, it is helpful if the Python window is fairly wide to make it easier to see longer lines of code.

You are almost ready to run some geoprocessing tools in the Python window. Before doing so, you next need to confirm some geoprocessing options.

7   On the ArcMap menu bar, click Geoprocessing > Geoprocessing Options.

8 On the Geoprocessing Options dialog box, make sure the "Overwrite the outputs of geoprocessing operations" and "Add results of geoprocessing operations to the display" check boxes are selected.
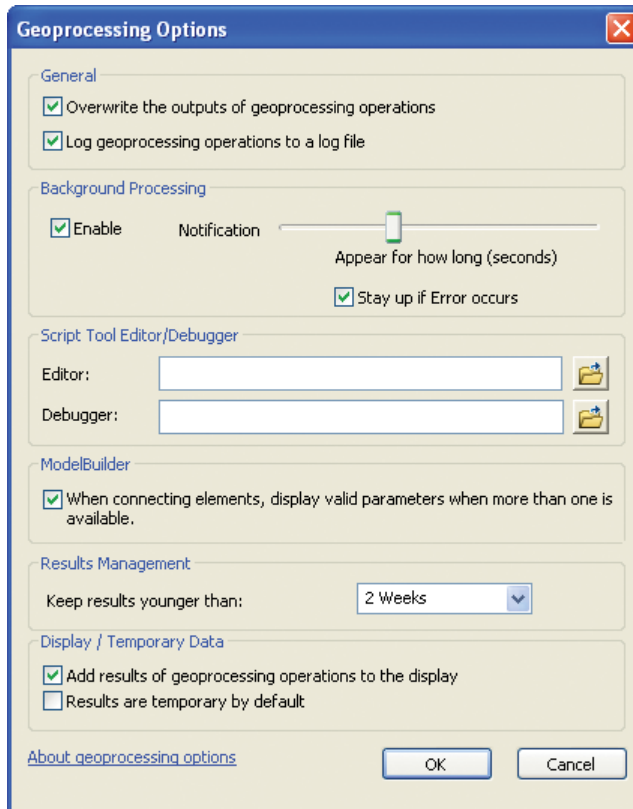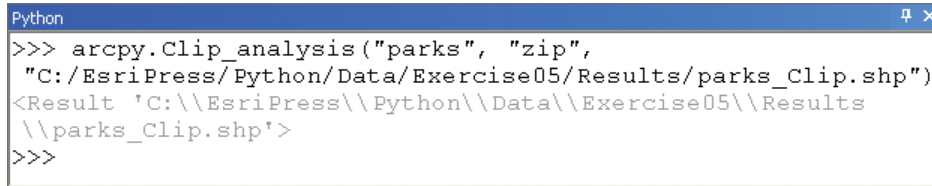

9 Click OK to close the Geoprocessing Options dialog box.

10 In the Python window, at the command prompt, enter the following code and press ENTER:

```
>>> arcpy.Clip_analysis("parks", "zip", "C:/EsriPress/Python/Data/
Exercise05/Results/parks_Clip.shp")
```

In the Python window, it is not necessary to start your code with the `import ArcPy` statement because the Python window is automatically aware of ArcPy. In a stand-alone script, however, the `import ArcPy` statement is needed to import the ArcPy site package, including all its modules and functions.

Note: If the exercise data was installed on a different drive or in a different folder, the path in the preceding code needs to be replaced by the correct path to the Results folder for exercise 5.

When you press ENTER after the last line of code, the Clip tool is run. Upon completion the newly created shapefile parks_Clip.shp is added as a layer to the data frame, and the result is printed to the Python window.

```
Python                                                    ▯ ×
>>> arcpy.Clip_analysis("parks", "zip",
 "C:/EsriPress/Python/Data/Exercise05/Results/parks_Clip.shp")
<Result 'C:\\EsriPress\\Python\\Data\\Exercise05\\Results
\\parks_Clip.shp'>
>>>
```

A few things to notice about the syntax:

- When your line of code exceeds the width of the Python window, simply keep typing and the line of code will wrap. It does not affect code execution.

- Using `arcpy.Clip_analysis` is the same as `arcpy.analysis.Clip`.

- When layers are added to the map document, they can be referenced by their layer name—in this case, parks. When datasets are referenced on disk, the file extension .shp is required—in this case, parks.shp. Unless the correct workspace is set, you need to reference datasets using the full path, such as C:\EsriPress\Python\Data\Exercise05\parks.shp. If the correct workspace is set and the dataset is in the workspace, it is not necessary to use a full path, and you need to use only the name of the dataset—that is, parks.shp.

- When referencing data on disk, you can limit the need to write the full path by setting the workspace as part of the environment properties.

11 **Run the following code:**

```
>>> from arcpy import env
```

The `env` class is now imported, making it possible to set environment properties.

12 **Run the following code:**

```
>>> env.workspace = "C:/EsriPress/Python/Data/Exercise05"
```

Running the code sets the current workspace to the folder containing the exercise data. You can now reference data on disk without having to type the full path each time.

Next, you will use another geoprocessing tool.

Note: Instead of first importing the `env` *class and then setting the current workspace, you can use a single line of code to do this:* `arcpy.env.workspace = ....`

13 Run the following code:

```
>>> arcpy.Buffer_analysis("facilities.shp", "Results/facilities_
 buffer.shp", "500 METERS")
```

The Buffer tool is run and the resulting shapefile is added to the data frame. By default, the Buffer tool creates a new feature around each input feature and the resulting buffers are allowed to overlap, as shown in the figure.
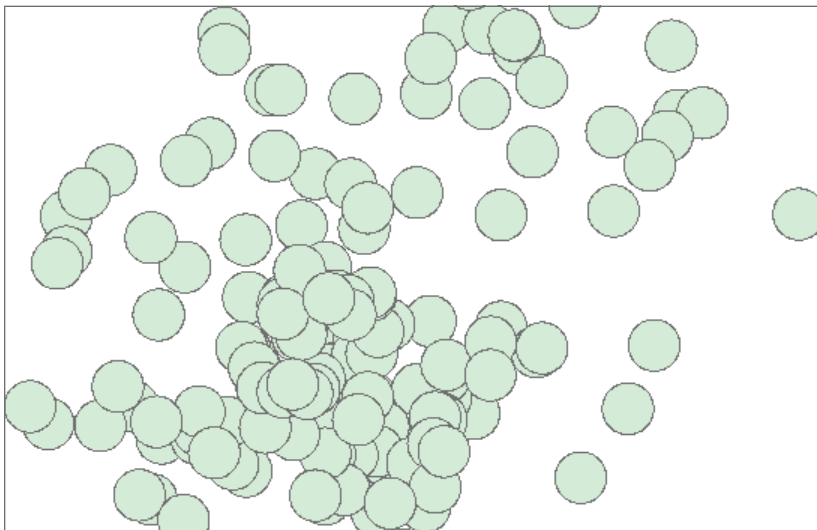
If you want these overlapping features to be dissolved, you need to set the Dissolve parameter. The syntax of the Buffer tool is as follows:

```
Buffer_analysis(in_features, out_feature_class,buffer_distance_
 or_field, {line_side}, {line_end_type},{dissolve_option},
 {dissolve_field})
```

Note: You can find this syntax by pressing F1 in the Python window or on the Help page for the Buffer tool.

The `dissolve_option` is an optional parameter for the Buffer tool. Because it is preceded by two optional parameters, you need to skip them using two empty strings ("", "").

14 At the prompt, use the UP ARROW key to bring up the previous line of code:

```
>>> arcpy.Buffer_analysis("facilities.shp", "Results/facilities_
buffer.shp", "500 METERS")
```

15 Next, modify this code to include the optional parameters:

```
>>> arcpy.Buffer_analysis("facilities.shp", "Results/facilities_
buffer.shp", "500 METERS", "", "", "ALL")
```

16 Press ENTER to run the code. All the buffer features are dissolved into a single multipart feature, as shown in the figure.
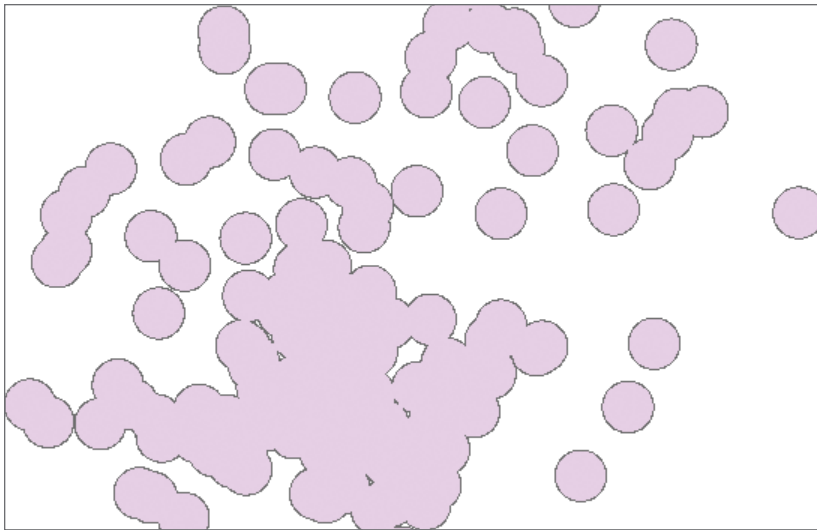
So far, the tool parameters have been hard-coded—that is, the actual values have been used. Alternatively, you can first assign the value of a parameter to a variable, and then use the variables in the code that calls the tool.

17 Run the following code:

```
>>> in_features = "bike_routes.shp"
>>> clip_features = "zip.shp"
>>> out_features = "bike_Clip.shp"
>>> xy_tolerance = ""
```

This creates a variable for each of the tool's parameters. Next, you are ready to run the tool.

18 **Run the following code:**

```
>>> arcpy.Clip_analysis(in_features, clip_features, out_features, ➜
➜ xy_tolerance)
```

When you press ENTER, the Clip tool is run. The use of variables is not required, but it gives your code more flexibility.

Get help with tool syntax

Working with geoprocessing tools in Python requires a good understanding of the syntax of the tools. The proper syntax can be reviewed in a number of ways.

1 **Make sure the Help and syntax panel is visible within the Python window.**

2 **At the command prompt, enter the following code (without pressing ENTER):**

```
>>> arcpy.Buffer_analysis(
```

Entering the code brings up the syntax for the tool.
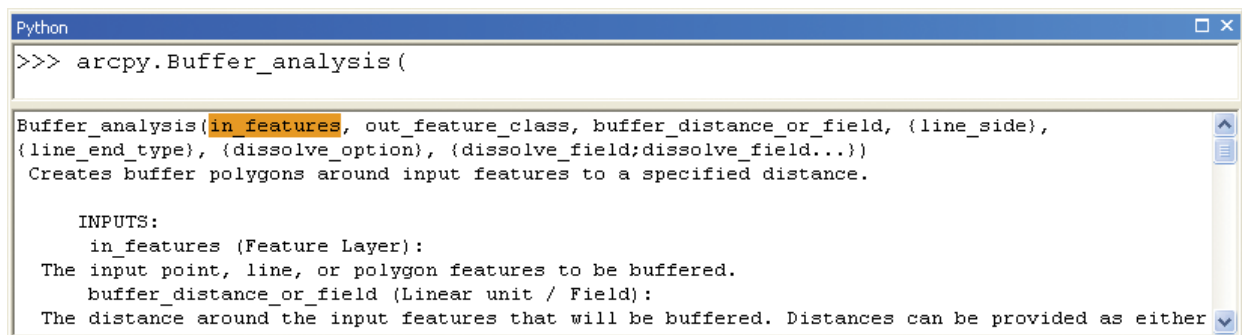
You can also find the syntax and tool explanation in the Item Description and Help files for each tool.

3 **On the ArcMap Standard toolbar, click the Search button to open the Search window.**

4 In the Search window, click Tools to filter the results. In the text box, type Buffer **and press the Search button** 🔍 **.**
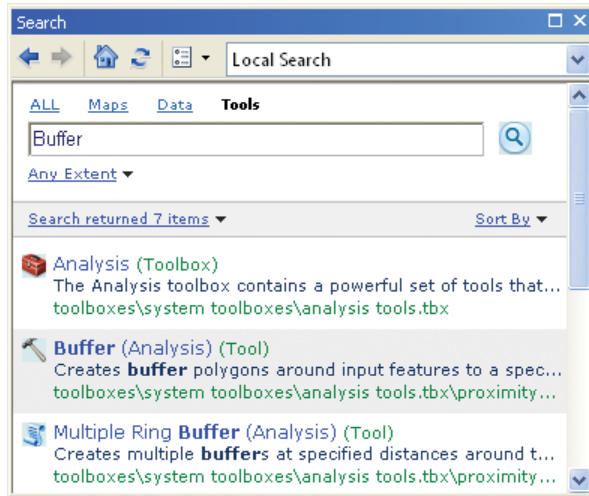
5 In the list of results, click the definition of the Buffer (Analysis) entry, starting with "Creates buffer polygons …". This opens the Item Description of the Buffer tool. It contains the same information as in the ArcGIS Desktop Help files.

Buffer (Analysis)

Title   Buffer (Analysis)

Summary

Creates buffer polygons around input features to a specified distance.

Illustration

6   Scroll down in the Item Description window to see the syntax of the tool.

Syntax

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

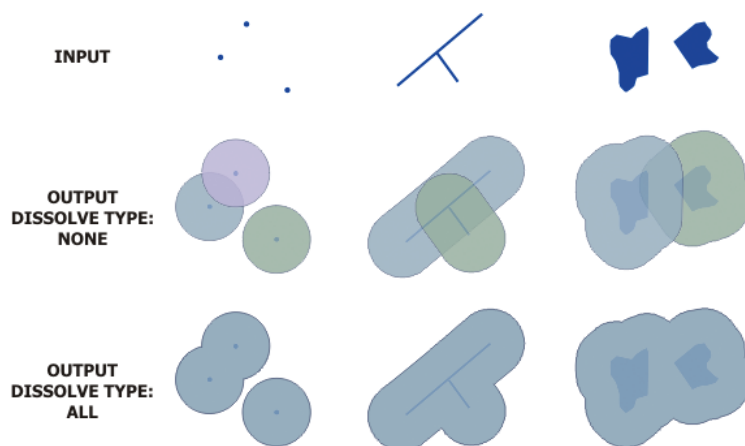| Parameter	Explanation	Data Type
in_features	**Dialog Reference** The input point, line, or polygon features to be buffered. **Python Reference** The input point, line, or polygon features to be buffered.	Feature Layer
out_feature_class	**Dialog Reference** The feature class containing the output buffers. **Python Reference** The feature class containing the output buffers.	Feature Class
buffer_distance_or_field	**Dialog Reference** 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

7   Scroll farther down in the Item Description window to see code examples under Code Samples.

Code Samples

Buffer example (Python window)

The following Python window script demonstrates how to use the Buffer tool.

```
import arcpy
arcpy.env.workspace = "C:/data"
arcpy.Buffer_analysis("roads", "C:/output/majorrdsBuffered", "100 Feet", "FULL", "ROUND", "LIST", "Distance")
```

Buffer example (stand-alone script)

Find areas of suitable vegetation that exclude areas heavily impacted by major roads:

```
# Name: Buffer.py
# Description: Find areas of suitable vegetation which exclude areas heavily impacted by major roads

# import system modules
import arcpy
from arcpy import env

# Set environment settings
env.workspace = "C:/data/Habitat_Analysis.gdb"

# Select suitable vegetation patches from all vegetation
veg = "vegtype"
suitableVeg = "C:/output/Output.gdb/suitable_vegetation"
whereClause = "HABITAT = 1"
arcpy.Select_analysis(veg, suitableVeg, whereClause)
```

Finally, code autocompletion prompts can help you write the proper syntax. For example, when you start typing the parameters for the Buffer tool, the Python window recognizes which parameters you are currently working on. In the case of the `dissolve_option` parameter, the options are "ALL", "LIST", and "NONE".

```
Python
>>> arcpy.Buffer_analysis("facilities", "Results/facilities_buffer.shp",
 "500 METERS","", "", "
                    "ALL"
                    "LIST"
                    "NONE"
```
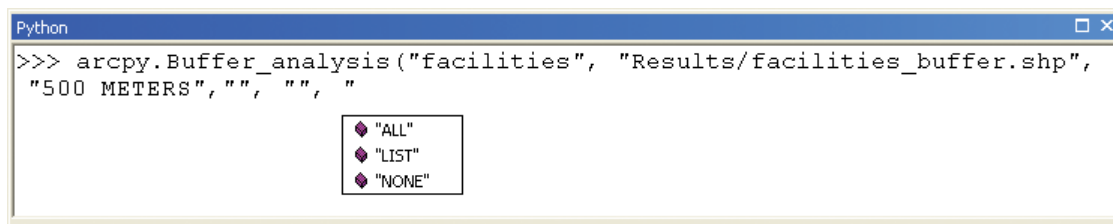
8   Close the Item Description window.

Explore ArcPy functions and classes

All the geoprocessing tools in ArcGIS are functions of ArcPy. There are also a number of ArcPy functions that are not tools.

Note: The instructions in this exercise assume you are completing all the steps in sequence without closing ArcMap. When ArcMap is closed, any code entered in the Python window is removed from memory. As a result, when restarting ArcMap, you may need to reenter portions of earlier code. In this case, the necessary code would consist of the following:

1   Run the following code:

```
>>> arcpy.Exists("hospitals.shp")
```

The result is True.

```
>>> from arcpy import env
>>> env.workspace = "C:/EsriPress/Python/Data/Exercise05"
```

Note: Unlike with ArcMap, closing the Python window does not eliminate the code. Moreover, using Clear or Clear All removes the code from the Python window, but the code that has already been run is still in memory. For example, once you import ArcPy and set the workspace, you do not have to do it again in the same ArcMap session, even if these lines of code are no longer visible.

The `Exists` function returns a Boolean value. There are several other ArcPy functions that are not geoprocessing tools, and some of them are used later in this exercise and other exercises.

One function, in particular, is a useful shortcut for getting the syntax of ArcPy functions. It is the `Usage` function, which you'll use next.

2   Run the following code:

```
>>> arcpy.Usage("Clip_analysis")
```

The result is `'Clip_analysis(in_features, clip_features, out_feature_class, {cluster_tolerance})\nExtracts input features that overlay clip features.'`

Remember to call geoprocessing tools using a toolbox alias. Calling a tool only by name will produce an error.

3 Run the following code:

```
>>> arcpy.Usage("Clip")
```

The result is `u'Method Clip not found. Choices: Method Clip not unique, please use ToolboxName_ToolName.'`

The `Usage` function applies to all ArcPy functions, and not just geoprocessing tools.
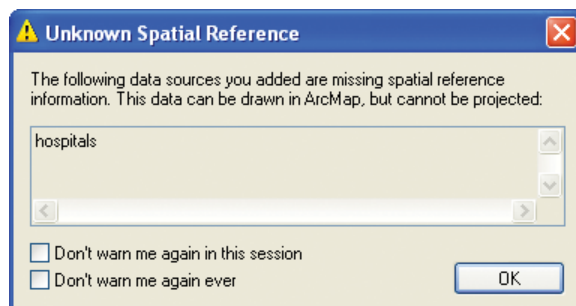
4 Run the following code:

```
>>> arcpy.Usage("Exists")
```

The result is `'exists(<dataset>, {datatype}) -> boolean\nCheck if a data element exists.'`

`Exists` is a function and not a tool, so it doesn't require a toolbox alias name as the Clip tool does.

In addition to functions, ArcPy also contains a number of classes. Classes are often used as shortcuts to complete tool parameters. You have already become familiar with using the `env` class to set environment properties. Another commonly used class is the `SpatialReference` class.

5 In ArcMap, drag the hospitals.shp layer into the current map document. The spatial reference of this shapefile is missing.

You will next use the Define Projection tool to fix it. The syntax of the Define Projection tool is

```
DefineProjection_management(in_dataset, coor_system)
```

6 Run the following code:

```
>>> prjFile = "C:/EsriPress/Python/Data/Exercise05/facilities.prj"
>>> spatial_ref = arcpy.SpatialReference(prjFile)
```

This spatial reference object can now be used for other purposes.

7 Run the following code:

```
>>> arcpy.DefineProjection_management("hospitals", spatial_ref)
```

The result is `<Result 'hospitals'>`.

The spatial reference object is used in the Define Projection tool to specify the coordinate system of the hospitals.shp file. In this example, the coordinate system parameter could also be specified by directly referencing the .prj file. However, creating a spatial reference object also gives you access to its many properties.

8 Run the following code:

```
>>> print spatial_ref.name
```

The result is

```
NAD_1983_StatePlane_Texas_Central_FIPS_4203_Feet
```

```
>>> print spatial_ref.linearUnitName
```

The result is `Foot_US`.

```
>>> print spatial_ref.XYResolution
```

The result is `0.000328083333333`.

Control the environment settings

You have already seen how to set the current workspace using the `env` class. This class contains many different environment settings that can be controlled using Python.

1   On the ArcMap menu bar, click Help > ArcGIS Desktop Help. On the Search tab, type `env` **and press ENTER. From the list of results, click env (arcpy) to view the Help page for this class.**

---

env (arcpy)

ArcGIS 10.1

Locate topic

Summary

Environment settings are exposed as properties on ArcPy's **env** class. These properties can be used to retrieve the current values or to set them. Geoprocessing environment settings can be thought of as additional parameters that affect a tool's results.

Properties

Property	Explanation	Data Type
autoCommit (Read and Write)	Tools that honor the Auto Commit environment will force a commit after the specified number of changes have been made within an ArcSDE transaction. Learn more about autoCommit.	Long
cartographicCoordinateSystem (Read and Write)	Tools that honor the Cartographic Coordinate System environment will use the specified coordinate system to determine the size, extent, and spatial relationships of features when making calculations. Learn more about cartographicCoordinateSystem.	String
cartographicPartitions (Read and Write)	Tools that honor the Cartographic Partitions environment will subdivide input features by the specified partition polygon features for sequential processing to avoid memory limitations that may otherwise be encountered with large datasets. Learn more about cartographicPartitions.	String
cellSize (Read and Write)	Tools that honor the Cell size environment setting set the output raster cell size, or resolution, for the operation. The default output resolution is determined by the coarsest of the input raster datasets. Learn more about cellSize.	String

---

In most cases, you do not have to worry about all these properties, just a few selected ones.

2   Close ArcGIS Desktop Help and return to the Python window.

3 Run the following code:

```
>>> env.overwriteOutput = True
```

Running the code enables overwriting the outputs of geoprocessing operations. So even if this property has not been set on the Geoprocessing Options dialog box in ArcMap, you can control it within a script.

4 Run the following code:

```
>>> env.outputCoordinateSystem = spatial_ref
```

The output coordinate system is set based on the spatial reference object defined earlier.

Note: As before, if for whatever reason you have to close ArcMap, none of the earlier lines of code will be kept in memory. When you start a new ArcMap session, your code would have to look as follows:

```
>>> from arcpy import env
>>> prj_file = "C:/EsriPress/Python/Data/Exercise05/facilities.prj"
>>> spatial_ref = arcpy.SpatialReference(prjFile)
>>> env.overwriteOutput = True
>>> env.outputCoordinateSystem = spatial_ref
```

The complete list of current environment settings can be obtained using the ListEnvironments function.

5 Run the following code:

```
>>> environments = arcpy.ListEnvironments()
>>> for environment in environments:
...     env_setting = eval("env." + environment)
...     print "{0}: {1}".format(environment, env_setting)
...
```

This example code uses the built-in Python eval function. The argument of the function is a Python expression. The eval function evaluates the expression and returns it as a value. In this case, the function returns a value that represents the particular environment setting.

The `for` loop iterates over the list of environment settings, and they are printed in the Python window:

```
newPrecision: SINGLE
autoCommit: 1000
XYResolution: None
XYDomain: None
...
workspace: C:/EsriPress/Python/Data/Exercise05
...
```

You can also clear specific environment settings or reset all values to their default.

6 Run the following code:

```
>>> arcpy.ClearEnvironment("workspace")
>>> print env.workspace
```

The result is

```
C:\Documents and Settings\<User>\My Documents\ArcGIS\Default.gdb
```

7 Run the following code:

```
>>> arcpy.ResetEnvironments()
>>> print env.outputCoordinateSystem
```

The result is `None`.

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

Work with tool messages

Messages are automatically written to the Results window when tools are run. You can also access these messages from within Python.

In the next set of steps, you will run a script from PythonWin. You can enter the same code in the Python window, but as code gets a bit longer, it is often easier to work with script files.

1 Start PythonWin.

2 On the PythonWin Standard toolbar, click the New button. Select Python Script and click OK.

3 In the script window, enter the following code:

```
import arcpy
from arcpy import env
env.workspace = "C:/EsriPress/Python/Data/Exercise05"
env.overwriteOutput = True
arcpy.Clip_analysis("parks.shp", "zip.shp", "Results/parks_Clip.shp")
print arcpy.GetMessages()
```

4 On the PythonWin Standard toolbar, click the Save button and save as my_clip.py **to the C:\EsriPress\Python\Data\Exercise05\Results folder.**

5 Click the Run button. On the Run Script dialog box, do not select a Debugging option and click OK. The Clip tool is run. At the bottom of the PythonWin interface, a message should appear on the status bar, like the example in the figure.

Script 'C:\EsriPress\Python\Data\Exercise05\Results\my_clip.py' returned exit code 0 | | NUM | | 00001 | 001

Exit code 0 means no errors were encountered.

6 Examine the contents of the Interactive Window in PythonWin. The Interactive Window should look something like the example in the figure. These are the same messages that are normally sent to the Results window.

Instead of printing all the messages, you can specify one in particular, as you'll do next.

7 Replace the last line of the code in your my_clip.py script with the following:

```
msgCount = arcpy.GetMessageCount()
print arcpy.GetMessage(msgCount-1)
```

8   Save and run your my_clip.py script. This code runs the Clip tool, determines the number of messages, and returns only the last message:

```
Succeeded at Wed Feb 01 13:01:41 2012 (Elapsed Time: 1.00 seconds)
```

Only the messages from the last tool executed are kept by ArcPy. To obtain messages after multiple tools are run, you can use a result object.

9   Modify your my_clip.py script as follows:

```
import arcpy
from arcpy import env
env.workspace = "C:/EsriPrss/Python/Data/Exercise05"
env.overwriteOutput = True
newclip = arcpy.Clip_analysis("bike_routes.shp", "parks.shp",
"Results/bike_Clip.shp")
fCount = arcpy.GetCount_management("Results/bike_Clip.shp")
msgCount = newclip.messageCount
print newclip.getMessage(msgCount-1)
```

10 Save and run your my_clip.py script. The script returns the last message from running the Clip tool, even though another tool was run after the Clip tool.

Work with licenses

When you import the ArcPy site package, you get access to all the geoprocessing tools in ArcGIS for Desktop. The tools that are included depend on the product level and the extensions that are installed and licensed.

1   Start ArcMap with a new blank document. Open the Python window.

2   At the command prompt, enter and run the following line of code:

```
>>> print arcpy.ProductInfo()
```

Running the code prints the current product license—for example, `arcview` or `arcinfo` (for ArcGIS for Desktop Basic and ArcGIS for Desktop Advanced, respectively). Alternatively, you can check whether a specific license is available.

3 At the command prompt, enter and run the following code:

```
>>> arcpy.CheckProduct("arcinfo")
```

If you are running ArcGIS for Desktop Advanced (`arcinfo`), the code
returns `AlreadyInitialized` or `Available`. If you are running a
lower license level, it returns `Unavailable`.

The same approach can be used to determine the availability of licenses
for extensions.

**4 On the ArcMap menu bar, click Customize >
Extensions.** The Extensions dialog box allows you to
select the extensions you want to use. ➤

**5 Select the check boxes for the extensions on the
list to see which ones have licenses available.** You
may get a warning message for some of them, like the
example in the figure.
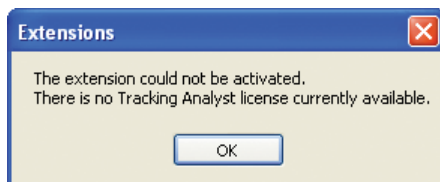
A warning means the extension has been installed but
not licensed. If you are able to turn on an extension
without getting a warning message, it means a license
was obtained.

For the purpose of the following steps, assume the
list of available licenses looks like the example in the
figure. ➤

Your list of available licenses may look somewhat dif-
ferent, so you may get different results when running
the code in the next set of steps.

6 Click Close to close the Extensions window.

7 Return to the Python window

8 Run the following code:

```
>>> arcpy.CheckExtension("tracking")
```
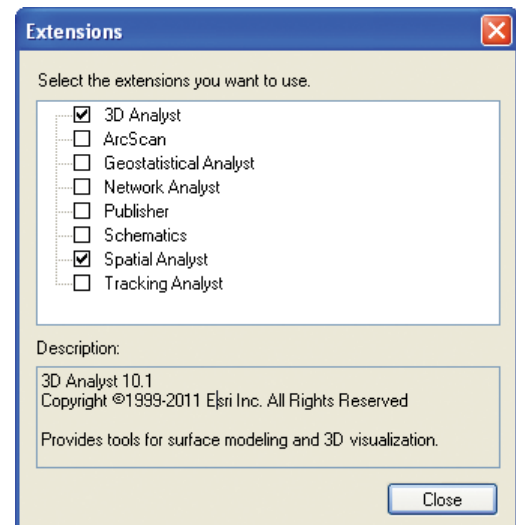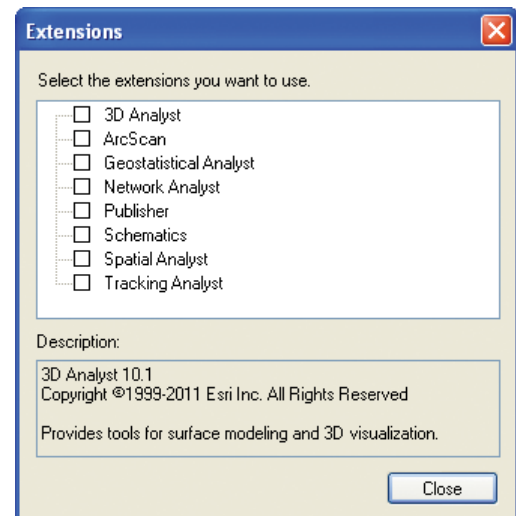
If a license is available, the resulting code is u'Available'. If no license is available, the resulting code is u'NotLicensed'.

9   Run the following code:

```
>>> arcpy.CheckExtension("spatial")
```

When working with geoprocessing tools, it is good practice to anticipate the licenses you will need. In the next set of steps, you will create a script that uses an ArcGIS for Desktop Advanced license.

10 Start PythonWin. Create a new Python script and save as centroid.py **to the Results folder for exercise 5.**

11 Enter the following lines of code:

```
import arcpy
from arcpy import env
env.workspace = "C:/EsriPress/Python/Data/Exercise05"
in_features = "parks.shp"
out_featureclass = "Results/parks_centroid.shp"
if arcpy.ProductInfo() == "ArcInfo":
    arcpy.FeatureToPoint_management(in_features, out_featureclass)
else:
    print "An ArcInfo license is not available."
```

12 Save and run the script. If you have an ArcGIS for Desktop Advanced license, the script runs the Feature to Point tool and creates a centroid for each feature in the input feature class. If you do not have an ArcGIS for Desktop Advanced license, the script generates a custom error message.

Note: When you import ArcPy, it automatically initializes the license based on the highest available license level.

A similar approach can be used when working with tools from the extensions. Licenses for extensions, however, need to be checked out and are not imported automatically with the import arcpy statement.

13 In PythonWin, create a new Python script and save as distance.py **to the Results folder for exercise 5.**

14 Enter the following lines of code:

```
import arcpy
from arcpy import env
env.workspace = "C:/EsriPress/Python/Data/Exercise05"
if arcpy.CheckExtension("spatial") == "Available":
    arcpy.CheckOutExtension("spatial")
    out_distance = arcpy.sa.EucDistance("bike_routes.shp", cell_size = ➜
100)

    out_distance.save("C:/EsriPress/Python/Data/Exercise05/Results/ ➜
bike_dist")
    arcpy.CheckInExtension("spatial")
else:
    print "Spatial Analyst license is not available."
```

15 Save and run the script. If you have an ArcGIS Spatial Analyst license, the script runs the Euclidean Distance tool and creates a new distance grid. If you do not have an ArcGIS Spatial Analyst license, the script generates a custom error message.

It is good practice to use the `CheckInExtension` function to return the license to the license manager when finished, so other applications can use it. However, all licenses are automatically returned to the license manager when a script is finished running.

Challenge exercises

Challenge 1

For each of the following tools, look up the syntax in ArcGIS Desktop Help and answer the following questions.

Tools:
- Add XY Coordinates
- Dissolve

Questions:
- What are the required parameters?
- What are the optional parameters, and what are their defaults?

Challenge 2

Write a script that runs the Add XY Features tool on the hospitals.shp feature class.

Challenge 3

Write a script that runs the Dissolve tool on the parks.shp feature class using the PARK_TYPE field as the field for aggregating features. Specify that multipart features are not allowed.

Challenge 4

Write a script that determines whether the following extensions are available: ArcGIS 3D Analyst, ArcGIS Network Analyst, and ArcGIS Spatial Analyst. The script should print an informative message with the results, such as `"The following extensions are available: ..."`.