

Exercise 5

Geoprocessing using Python

Exercise data

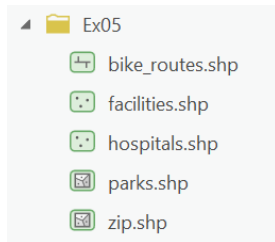
Exercise data for this book can be downloaded from

links.esri.com/PythonPro3rdEditionData . This is a link to the ArcGIS® Online group called *Python Scripting for ArcGIS Pro – 2024 (Esri Press)*. The data for exercise 5 is posted as a zip file called `PythonScripting_Ex05_Data.zip`. Download this file and extract it to a folder of your choice. The instructions use a folder called `C:\PythonPro`, but you can use a different folder provided you update any paths.

Use tools

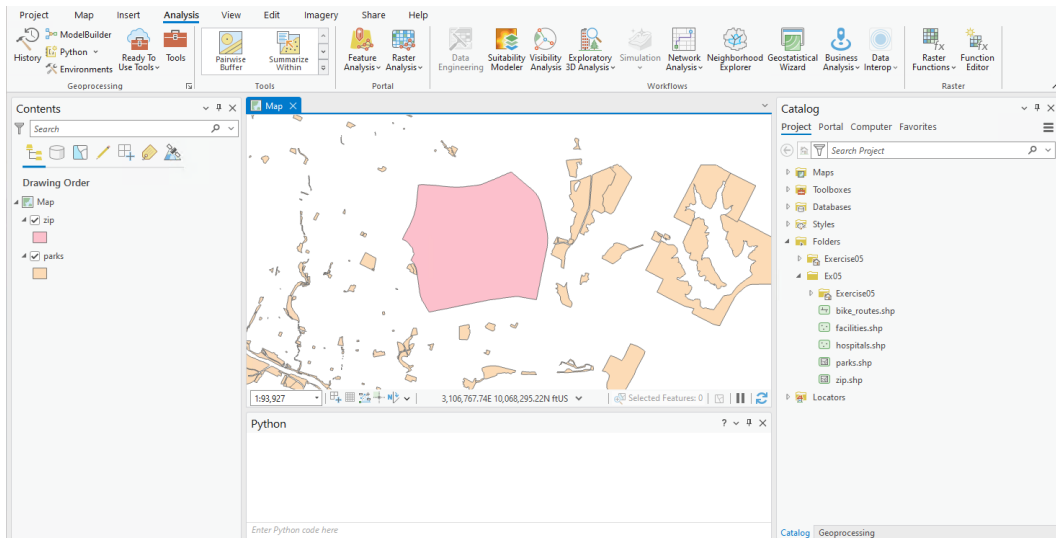
Before starting to work with the exercise data, you will preview the data in ArcGIS® Pro. When starting a new project in ArcGIS Pro, you can leave the home folder, default geodatabase, and default toolbox to their original settings or use the exercise folder—e.g., `C:\PythonPro\Ex05`). Saving your work is optional.

- 1. Start ArcGIS Pro and click Start Without a Template to create a new blank project.**
- 2. Make sure the Catalog pane is visible by clicking Catalog Pane on the View tab.
Dock the Catalog pane to the right side of the ArcGIS Pro interface.**
- 3. Create a new folder connection to the location of the exercise data by right-clicking Folders > Add Folder Connection and navigating to the folder—e.g., `C:\PythonPro\Ex05`.**
- 4. Examine the contents of this folder.**



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

5. **Add the parks.shp and zip.shp files to a new map. You can remove the default basemap from the map display.**
6. **Bring up the Python window by clicking Python on the Analysis tab.**
7. **Dock the Python window at the bottom of the ArcGIS Pro interface.**

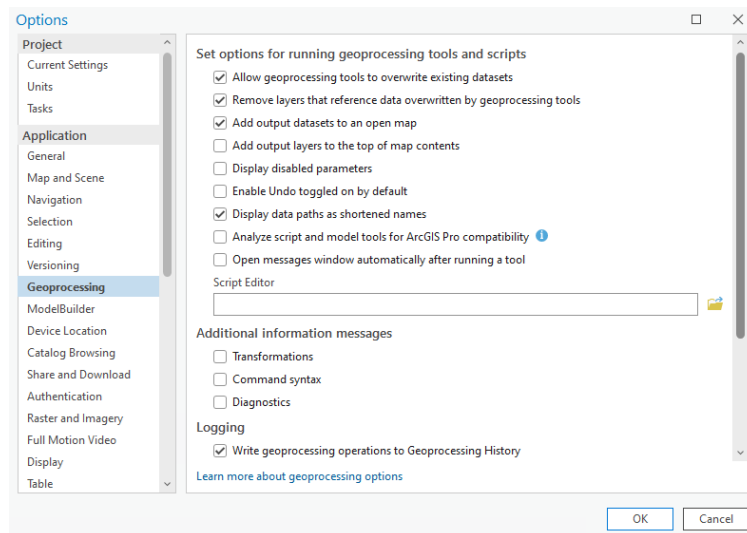


Note: Docking the Catalog pane and Python window is not required, but it helps organize your available desktop space. Typically, it is helpful if the Python window is wide enough to make it easier to see longer lines of code.

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

8. On the Analysis tab, click Geoprocessing Options, located in the lower-right corner of the Geoprocessing group.

9. In the Options dialog box, in the Geoprocessing panel, make sure the four options shown are checked.



10. Click OK to close the Options dialog box.

11. In the Python window, at the prompt, enter the following code and press

Enter:

```
arcpy.Clip_analysis("parks", "zip",  
"C:/PythonPro/Ex05/parks_clip.shp")
```

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

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 package, including all its modules, classes, and functions.

When you press Enter after the last line of code, the Clip tool runs. Upon completion, the newly created shapefile, `parks_clip.shp`, is added as a layer to the active map, and the result is printed to the Python window.

A screenshot of the Python window in ArcGIS. The window has a title bar with the word "Python" and standard window controls. The main area shows a single line of code: `arcpy.Clip_analysis("parks", "zip", "C:/PythonPro/Ex05/parks_clip.shp")`. Below the code, the output is displayed: `<Result 'C:\\PythonPro\\Ex05\\parks_clip.shp'>`.

```
Python
arcpy.Clip_analysis("parks", "zip", "C:/PythonPro/Ex05/parks_clip.shp")
<Result 'C:\\PythonPro\\Ex05\\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 using `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 must

reference datasets using the full path, such as C:\PythonPro\Ex05\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 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.

12.Run the following code:

```
arcpy.env.workspace = "C:/PythonPro/Ex05"
```

Running this code sets the current workspace to the folder containing the exercise data.

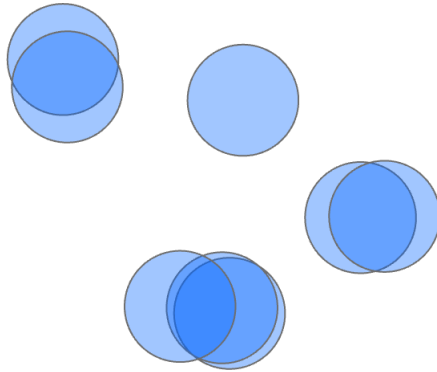
You can now reference data on disk without having to type the full path each time.

Next, you will use another geoprocessing tool.

13.Run the following code:

```
arcpy.Buffer_analysis("facilities.shp",  
"facilities_buffer.shp", "500 METERS")
```

The Buffer tool runs, 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 can overlap, as shown in the figure.



If you want these overlapping features to be dissolved, you must 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 look up this syntax on the help page for the Buffer tool, but it also appears in the Python window. When you type `arcpy.Buffer_analysis(`, the syntax for the tool appears as a pop-up.

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

14. At the prompt, use the up arrow to bring up the previous line of code:

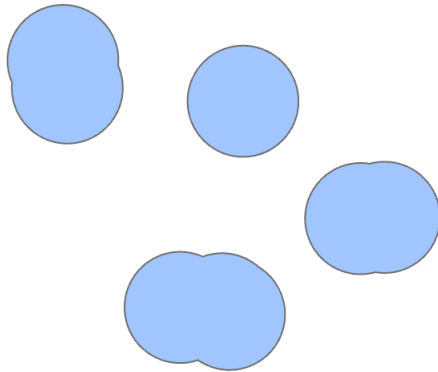
```
arcpy.Buffer_analysis("facilities.shp",  
"facilities_buffer.shp", "500 METERS")
```

15. Modify this code to include the optional parameters:

```
arcpy.Buffer_analysis("facilities.shp",  
"facilities_buffer.shp", "500 METERS", "", "", "ALL")
```

Press Enter to run the code.

All the buffer features are dissolved into a single feature, like the example 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.

16. Run the following code, one line at a time:

```
in_features = "bike_routes.shp"  
  
clip_features = "zip.shp"  
  
out_features = "bike_clip.shp"  
  
xy_tolerance = ""
```

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

17. Run the following code:

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

When you press Enter, the Clip tool runs, and the new feature class, bike_clip.shp, is added as a layer to the active map. The use of variables is not required, but it gives your code more flexibility.

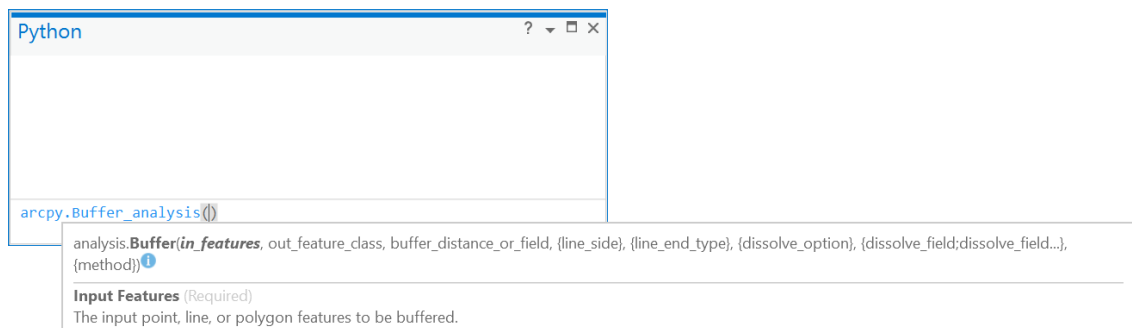
Get help with tool syntax

Working with geoprocessing tools in Python requires a good understanding of the syntax of the tools. The proper syntax can be reviewed in several ways. The quickest way to views the syntax is directly in the Python window.

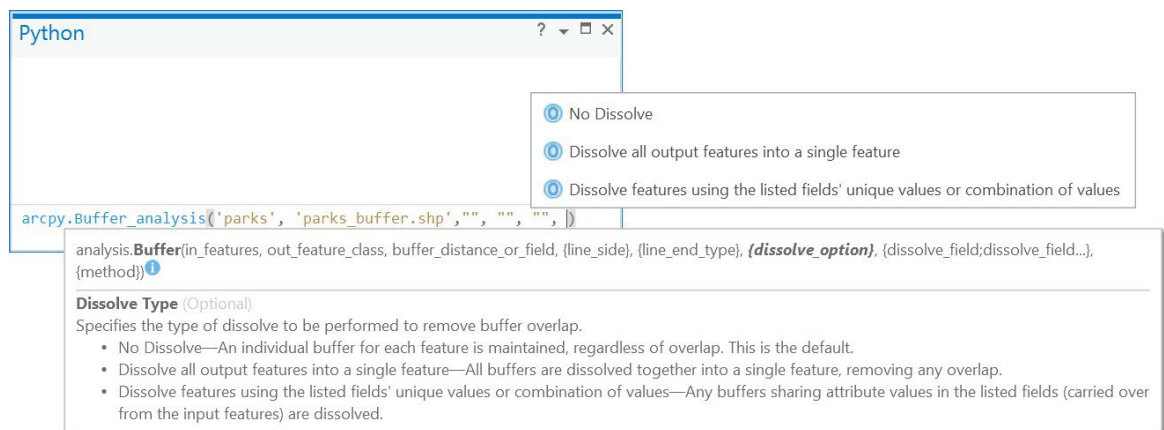
1. At the prompt, enter the following code (without pressing Enter):

```
arcpy.Buffer_analysis(
```

Entering the code brings up the syntax for the tool.



As you type the parameters for a tool, the active parameter is shown in bold italic, which makes it easier to follow along with the syntax as you type the code. Code autocompletion prompts also 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, three options appear to select from, with a more detailed explanation of each option below the syntax.

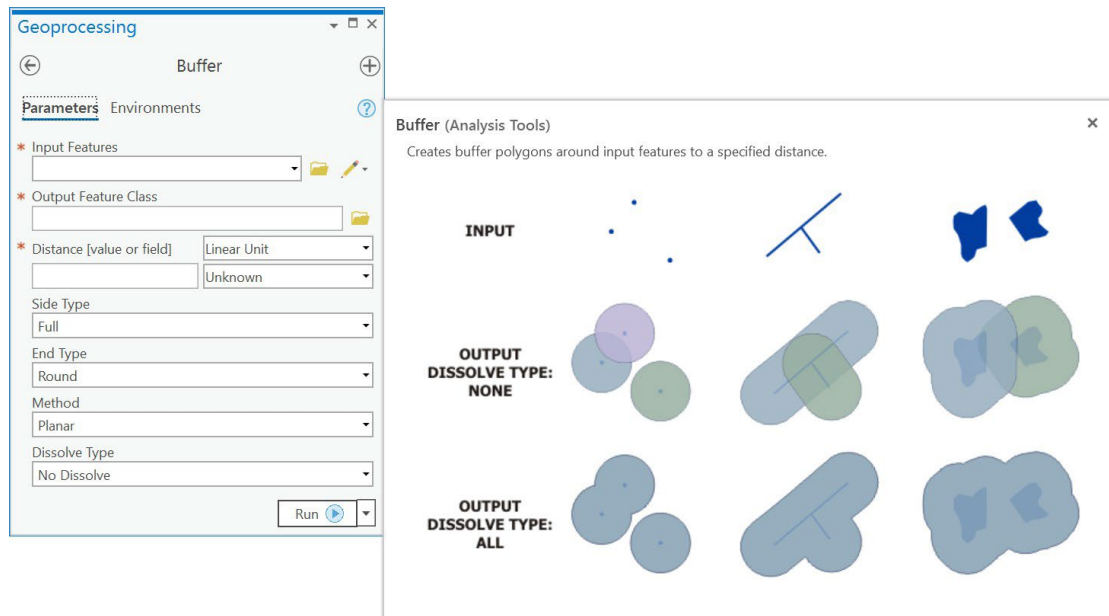


Note that the options that appear on-screen are "plain English" explanations of the parameter values, not the actual values used in code. For example, in the case of the `dissolve_option` parameter, the three options are `NONE`, `ALL`, and `LIST`.

You can also find the syntax and tool explanation on the help page for each tool.

2. On the Analysis tab, click Tools to open the Geoprocessing pane.
3. In the Find Tools box, type **Buffer**.
4. In the list of results, click on the Buffer tool to open the Buffer tool dialog box.

5. Hover your cursor over the Help icon (blue question mark in a light-blue circle) in the upper-right corner of the tool dialog box, and a pop-up window appears with a brief explanation of the tool.



6. Click on the Help icon to open in your default web browser the online help page for the Buffer tools.

You can also find the help pages for any tool by searching for the name of the tool on the general ArcGIS Pro help page: <https://pro.arcgis.com/en/pro-app/help>.

Explore ArcPy functions and classes

All the geoprocessing tools in ArcGIS Pro are functions of ArcPy. There are also several ArcPy functions that are not tools.

1. Run the following code:

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

The result is True.

You may need to reenter portions of earlier code. In this case, the necessary code would consist of the following:

```
arcpy.env.workspace = "C:/PythonPro/Ex05"
```

Note: Closing the Python window does not remove the code. Moreover, using Clear Transcript removes the code from the Python window, but the code that has already been run is still in memory. For example, once you set the workspace, you normally do not have to do it again in the same session, even if these lines of code are no longer visible.

The `Exists()` function returns a Boolean value. Several other ArcPy functions also are not geoprocessing tools, and some of them are used later in this exercise and others.

The `Usage()` function is a useful shortcut for getting the syntax of ArcPy functions without relying on the pop-ups in the Python windows.

2. Run the following code:

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

The result is:

```
'Clip_analysis(in_features, clip_features,  
out_feature_class, {cluster_tolerance})'
```

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

3. Run the following code:

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

The result is

```
'Method Clip not found. Choices: Method Clip not found.'
```

The Usage function applies to all ArcPy functions, not only to geoprocessing tools.

4. Run the following code:

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

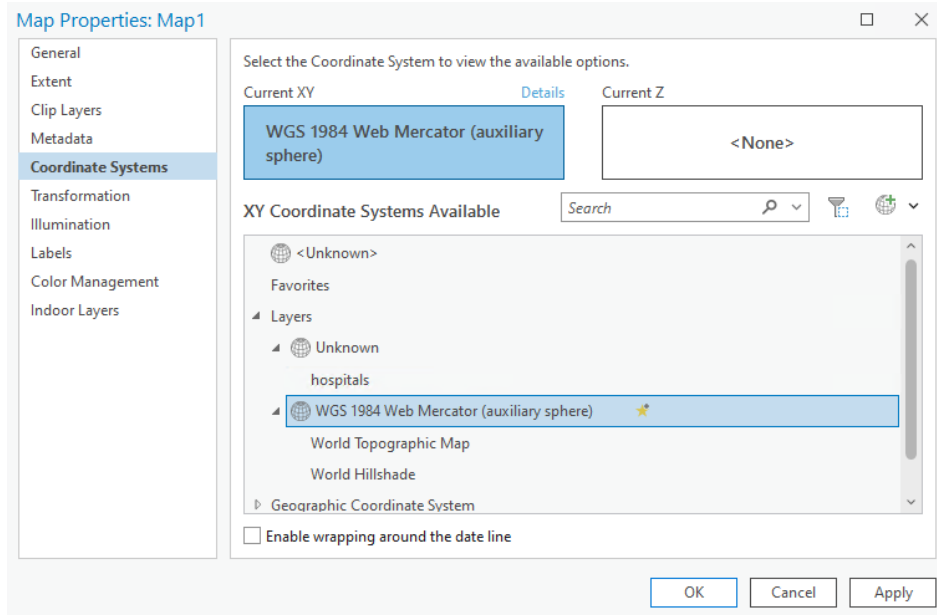
The result is

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

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

5. In ArcGIS Pro, add hospitals.shp to a new map.

The spatial reference of this shapefile is missing. As a result, the hospitals show up in northern Finland instead of Austin, Texas. This happens because the coordinates are displayed in the default Web Mercator coordinate system.



You will use the Define Projection tool to fix it. The syntax of the Define Projection tool is as follows:

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

6. Run the following code:

```
prjfile = "C:\\PythonPro\\Ex05\\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 `SpatialReference` 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 also could be specified by directly referencing the .prj file as a parameter of the Define Projection tool. 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.
```

9. Run the following code:

```
print(spatial_ref.linearUnitName)
```

The result is

```
Foot_US.
```

10. Run the following code:

```
print(spatial_ref.XYResolution)
```

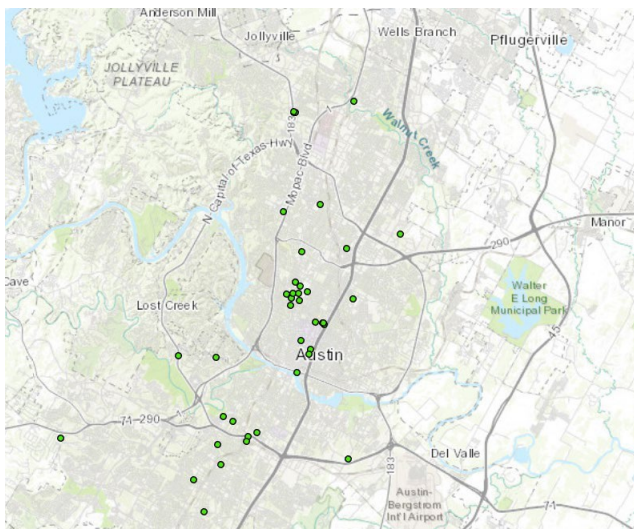
The result is

```
0.0003280833333333.
```

The result of running the Define Project tool is that the feature class hospitals.shp is assigned the correct coordinate system.

11. In the Content panel of the active map, right-click the hospitals data layer and click Zoom To Layer.

The hospitals are now showing where they belong—i.e., in Austin, Texas.



Control the environment settings

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

Several other properties are of interest.

1. Run the following code in the Python window:

```
arcpy.env.overwriteOutput = True
```

Running this code enables overwriting the outputs of geoprocessing operations. So even if this property has not been set as part of the geoprocessing options, you can control it within a script.

2. Run the following code:

```
arcpy.env.outputCoordinateSystem = spatial_ref
```

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

As before, if for whatever reason you closed ArcGIS Pro, your earlier code may not have been kept in memory unless it was saved as part of the project. When you start a new empty project, your code would have to look as follows:

```
arcpy.env.workspace = "C:/PythonPro/Ex05"
```

```
prjfile = "C:\\PythonPro\\Ex05\\facilities.prj"
```

```
spatial_ref = arcpy.SpatialReference(prjfile)
```

```
arcpy.env.overwriteOutput = True
```

```
arcpy.env.outputCoordinateSystem = spatial_ref
```

You can always check the value for a specific environment setting.

3. Run the following code:

```
print(arcpy.env.workspace)
```

The result is

```
'C:/PythonPro/Ex05'
```

4. Run the following code:

```
print(arcpy.env.outputCoordinateSystem.name)
```

The result is

```
'NAD_1983_StatePlane_Texas_Central_FIPS_4203_Feet'
```

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

5. Run the following code:

```
arcpy.ClearEnvironment("workspace")
```

```
print(arcpy.env.workspace)
```

The result is a default geodatabase, which looks like the following:

```
C:\PythonPro\Ex05\Exercise05\Default.gdb
```

Whatever the specific workspace that is printed, the workspace is no longer just

C:/PythonPro/Ex05.

6. Run the following code:

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

The result is 'None'.

Because all the environment settings have been reset to their default values, there is no longer a specific coordinate system used for the outputs of geoprocessing tools.

7. Close ArcGIS Pro.

There is no need to save your project.

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 IDLE. You can enter the same code in the Python window, but as code gets longer, it is often easier to work with script files.

1. **Start IDLE using the default environment or a clone.**
2. **On the top menu, click File > New File.**
3. **In the new script window, enter the following code:**

```
import arcpy

arcpy.env.workspace = "C:/PythonPro/Ex05"

arcpy.env.overwriteOutput = True

arcpy.Clip_analysis("parks.shp", "zip.shp",
"parks_clip.shp")

print(arcpy.GetMessages())
```

4. On the top menu of the script window, click File > Save As, and save as [my_clip.py](#) to the C:\PythonPro\Ex05 folder.

5. On the top menu of the script window, click Run > Run Module.

When the script runs, the Python Shell window opens, and a message is printed to confirm that the my_clip.py script was started. Following are messages to confirm that the tool ran successfully. These are the same messages that are normally encountered in ArcGIS Pro when running a geoprocessing tool.

```
>>> = RESTART: C:/PythonPro/Ex05/my_clip.py
Start Time: Tuesday, May 7, 2024 2:05:33 PM
Reading Features...
Cracking Features...
Assembling Features...
Succeeded at Tuesday, May 7, 2024 2:05:33 PM (Elapsed Time:
0.23 seconds)
```

Instead of printing all the messages, you can specify a single message, as you'll do next.

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

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

7. Save and run the my_clip.py script.

This code runs the Clip tool, determines the number of messages, and returns only the last message:

```
Succeeded at Tuesday, May 7, 2024 2:09:55 PM (Elapsed Time:  
0.20 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.

8. Modify your my_clip.py script as follows:

```

import arcpy

arcpy.env.workspace = "C:/PythonPro/Ex05"

arcpy.env.overwriteOutput = True

newclip = arcpy.Clip_analysis("bike_routes.shp",
"parks.shp", "bike_clip.shp")

fcount = arcpy.GetCount_management("bike_clip.shp")

msgCount = newclip.messageCount

print(newclip.getMessage(msgCount-1))

```

9. 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 ArcPy, you get access to all the geoprocessing tools in ArcGIS Pro. The tools that are included depend on the product level and the extensions that are installed and licensed.

1. **Start ArcGIS Pro and create a new blank project.**
2. **Open the Python window.**
3. **At the prompt, enter and run the following line of code:**

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

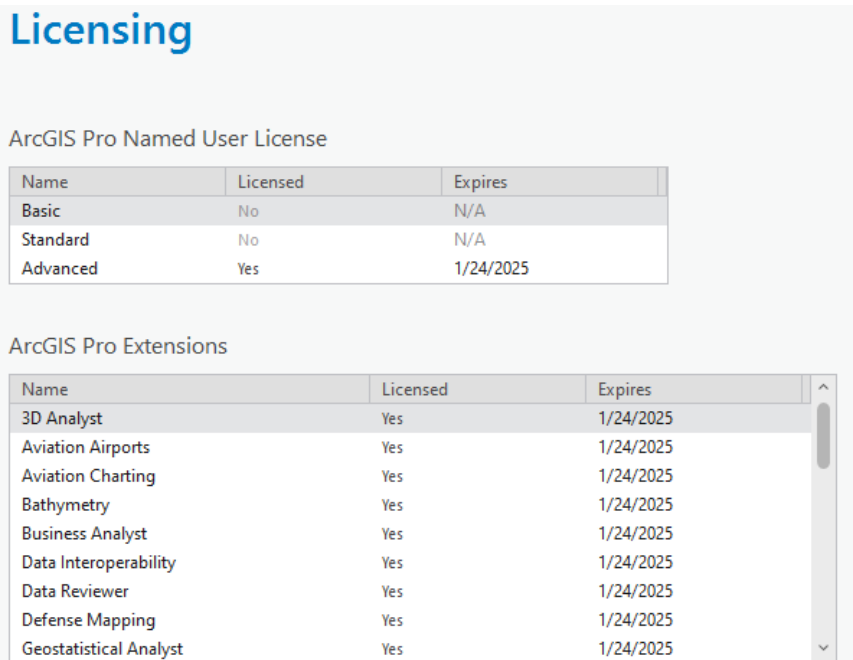
Running the code prints the current product license—for example, ArcInfo for an Advanced license. Alternatively, you can check whether a specific license is available.

4. At the prompt, enter and run the following code:

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

If you are running ArcGIS Pro with an Advanced license, 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.

You can check your licensing on the Project tab by clicking Licensing. For the purpose of the following steps, assume the licensing looks like the example in the figure.



The screenshot shows the 'Licensing' window in ArcGIS Pro. It has a title bar 'Licensing' in blue. Below the title bar, there is a section 'ArcGIS Pro Named User License' containing a table with three columns: 'Name', 'Licensed', and 'Expires'. The table lists three license types: 'Basic' (Not licensed, expires N/A), 'Standard' (Not licensed, expires N/A), and 'Advanced' (Licensed, expires 1/24/2025). Below this, there is a section 'ArcGIS Pro Extensions' containing a table with three columns: 'Name', 'Licensed', and 'Expires'. This table lists ten extensions, all of which are licensed and expire on 1/24/2025. The extensions are: 3D Analyst, Aviation Airports, Aviation Charting, Bathymetry, Business Analyst, Data Interoperability, Data Reviewer, Defense Mapping, and Geostatistical Analyst. The table has a scrollbar on the right side.

| Name | Licensed | Expires |
|----------|----------|-----------|
| Basic | No | N/A |
| Standard | No | N/A |
| Advanced | Yes | 1/24/2025 |

| Name | Licensed | Expires |
|------------------------|----------|-----------|
| 3D Analyst | Yes | 1/24/2025 |
| Aviation Airports | Yes | 1/24/2025 |
| Aviation Charting | Yes | 1/24/2025 |
| Bathymetry | Yes | 1/24/2025 |
| Business Analyst | Yes | 1/24/2025 |
| Data Interoperability | Yes | 1/24/2025 |
| Data Reviewer | Yes | 1/24/2025 |
| Defense Mapping | Yes | 1/24/2025 |
| Geostatistical Analyst | Yes | 1/24/2025 |

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

5. Return to the Python window.

6. Run the following code:

```
arcpy.CheckExtension("3d")
```

The result is 'Available'.

7. Run the following code:

```
arcpy.CheckExtension("business")
```

The result is 'NotLicensed'.

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 Pro Advanced license.

8. Open IDLE. Create a new Python script, and save it as [centroid.py](#) to the C:\PythonPro\Ex05 folder.

9. Enter the following lines of code:

```

import arcpy

arcpy.env.workspace = "C:/PythonPro/Ex05"

in_fc = "parks.shp"

out_fc = "parks_centroid.shp"

if arcpy.ProductInfo() == "ArcInfo":

    arcpy.FeatureToPoint_management(in_fc, out_fc)

else:

    print("An ArcInfo license is not available.")

```

10. Save and run the script.

If you have an ArcGIS for Advanced license, the script runs the Feature To Point tool and creates a centroid for each feature in the input feature class. You can make a folder connection to the Ex05 folder and view the newly created shapefile, parks_centroid.shp. If you do not have an ArcGIS Pro Advanced license, the script does not run the tool but generates a custom message.

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

End of Exercise 5.