Part 1
Learning the fundamentals of Python and geoprocessing

Exercise 1
Introducing Python

Install the exercise data

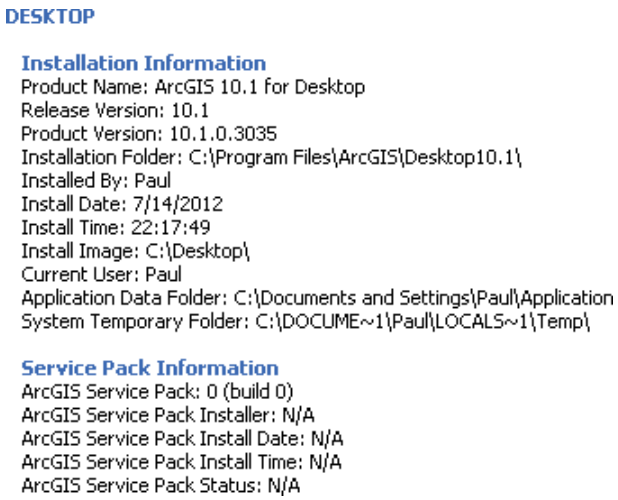
Exercise data is provided on the Data and Exercises DVD. Instructions on copying the exercise data from the DVD to a hard drive are included in appendix C. The default path is C:\EsriPress\Python, but you can change this during installation.

Check ArcGIS settings

You will first determine the version of the ArcGIS for Desktop software that is installed on your computer, including any available extensions.

1 **Close any ArcGIS for Desktop programs you may have open (the ArcMap application, the ArcCatalog application, and so on).**

2 **On the taskbar, click the Start button, and then on the Start menu, click All Programs > ArcGIS > ArcGIS Administrator.** This brings up the ArcGIS configuration information, which should read something like the figure. ➔



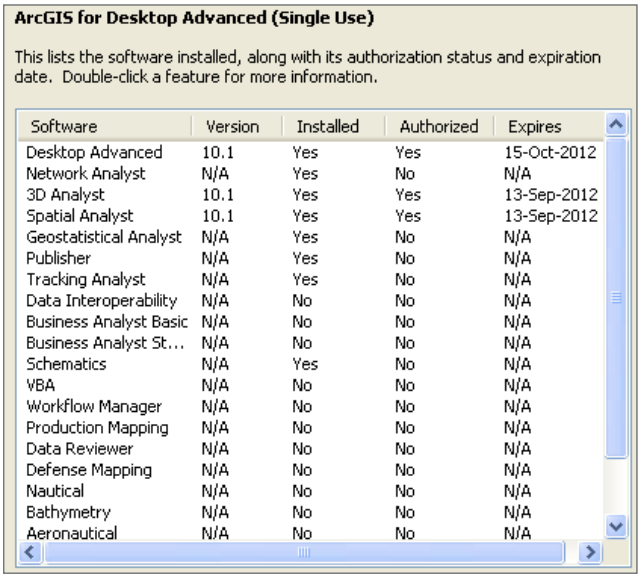
In this example, the version of ArcGIS for Desktop is 10.1, which is the same version used for the exercises in this book. If you are using version 10.1 with a recent service pack, there should be no noticeable difference in how the Python code works. If you are using version 10.0, some of the Python code in the

exercises will not work correctly because additional Python functionality has been introduced in 10.1.

If you are using ArcGIS for Desktop 9.3.1 or earlier, the code in the exercises will not work. The ArcPy site package was introduced in version 10.0, and this includes Python functionality that was not available in earlier versions.

Next, you will determine the license level and any available licenses.

3 **On the ArcGIS Administrator dialog box, click Desktop > Availability.** This brings up a list of the available licenses, which should look something like the figure. ➔



Software	Version	Installed	Authorized	Expires
Desktop Advanced	10.1	Yes	Yes	15-Oct-2012
Network Analyst	N/A	Yes	No	N/A
3D Analyst	10.1	Yes	Yes	13-Sep-2012
Spatial Analyst	10.1	Yes	Yes	13-Sep-2012
Geostatistical Analyst	N/A	Yes	No	N/A
Publisher	N/A	Yes	No	N/A
Tracking Analyst	N/A	Yes	No	N/A
Data Interoperability	N/A	No	No	N/A
Business Analyst Basic	N/A	No	No	N/A
Business Analyst St...	N/A	No	No	N/A
Schematics	N/A	Yes	No	N/A
VBA	N/A	No	No	N/A
Workflow Manager	N/A	No	No	N/A
Production Mapping	N/A	No	No	N/A
Data Reviewer	N/A	No	No	N/A
Defense Mapping	N/A	No	No	N/A
Nautical	N/A	No	No	N/A
Bathymetry	N/A	No	No	N/A
Aeronautical	N/A	No	No	N/A

The first entry shows the product level—that is, ArcGIS for Desktop Basic, ArcGIS for Desktop Standard, or ArcGIS for Desktop Advanced, formerly known as ArcView, ArcEditor, and ArcInfo, respectively. Whatever your product level, the code in the exercises will work correctly.

The following entries show the extensions that are installed and licensed. The only required extension for these exercises is Spatial Analyst, which is used in exercise 9. If you do not have ArcGIS Spatial Analyst installed and licensed, the code in exercise 9 will not work.

__Note:__ If you do not have the ArcGIS Spatial Analyst extension, you may want to consider installing the evaluation software provided with this book because it includes ArcGIS Spatial Analyst.

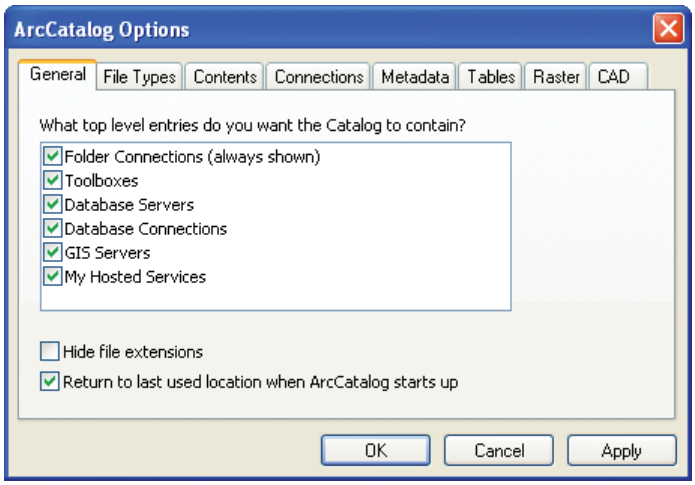
Next, you will select one of the options in ArcCatalog to make it easier to recognize file types.

4 **Close ArcGIS Administrator.**

5 **On the taskbar, click the Start button, and then, on the Start menu, click All Programs > ArcGIS > ArcCatalog 10.1.**

6 **On the ArcCatalog menu bar, click Customize > ArcCatalog Options.**

7 **On the General tab, make sure the "Hide file extensions" check box is cleared.** ➔

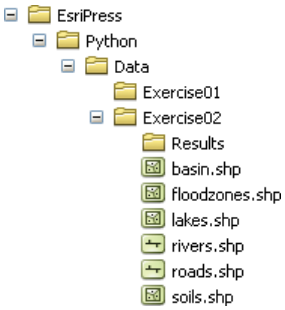


8 **Click OK to close the ArcCatalog Options dialog box.**

>>> TIP

To make it easier to work with the exercise data in this book, you can create a connection to the data folder. In the Catalog window, click the Connect to Folder button, browse to C:\EsriPress\Python\Data, and click OK. The shortcut will now appear under Folder Connections in the Catalog tree.

9 **Browse to where the data for the exercises is installed—that is, C:\EsriPress\Python\Data. Then open one of the folders that has data files in it—for example, Exercise02.** ➔



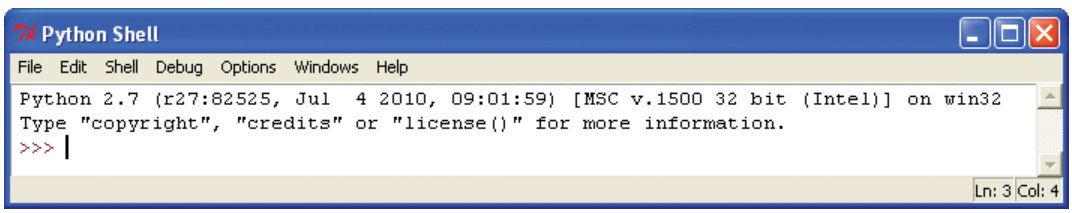
Notice that the files are now shown with their file extension, such as *basin.shp* instead of just *basin*.

10 **Close ArcCatalog.**

Check the Python version

You will now determine what version of Python is installed on your computer.

1 **On the taskbar, click the Start button, and then, on the Start menu, click All Programs > ArcGIS > Python 2.7 > IDLE (Python GUI).** This brings up the default Python editor, as shown in the figure.



ArcGIS 10.1 is designed to work with Python 2.7. At the final release of ArcGIS 10.1, the most recent production version of Python 2.7 was 2.7.2, which is the version that is installed as part of a typical ArcGIS 10.1 installation. Versions of Python 2.7 will continue to evolve, but the changes between these versions are so minor that ArcGIS 10.1 is expected to work with all versions of Python 2.7. The code in the exercises will generally also work fine for other 2.*x* versions of Python, including 2.6. However, when working with ArcGIS for Desktop, it is recommended that you run the version of Python that is installed with ArcGIS for Desktop. The code in the exercises will not work with Python 3.*x*.

Note: *It is possible to have multiple versions of Python on the same Windows computer. If you already have a different version of Python installed in addition to the one that is installed with ArcGIS for Desktop, be sure to use the version that is installed with ArcGIS for Desktop to write and run the code in these exercises.*

2 Close IDLE.

Install PythonWin

The exercises in the book use PythonWin as the editor application to work with Python. The following steps walk you through the installation of PythonWin.

When you install PythonWin, it determines the version of Python that is installed on your computer. The installation of ArcGIS 10.1 installs Python version 2.7.2. The PythonWin installation then assumes you have a program folder called Python 2.7—however, the ArcGIS 10.1 installation creates a program folder called ArcGIS\Python 2.7. Therefore, you first need to create a program folder simply called Python 2.7.

1 *For Windows XP:* **On the taskbar, right-click the Start button, and then click Open All Users.**

For Windows 7: **On the taskbar, click the Start button, and then, on the Start menu, right-click All Programs and click Open All Users.**

2 Double-click the Programs folder.

3 Right-click in the Programs folder and click New > Folder. Name the folder Python 2.7**. Close the Programs window.**

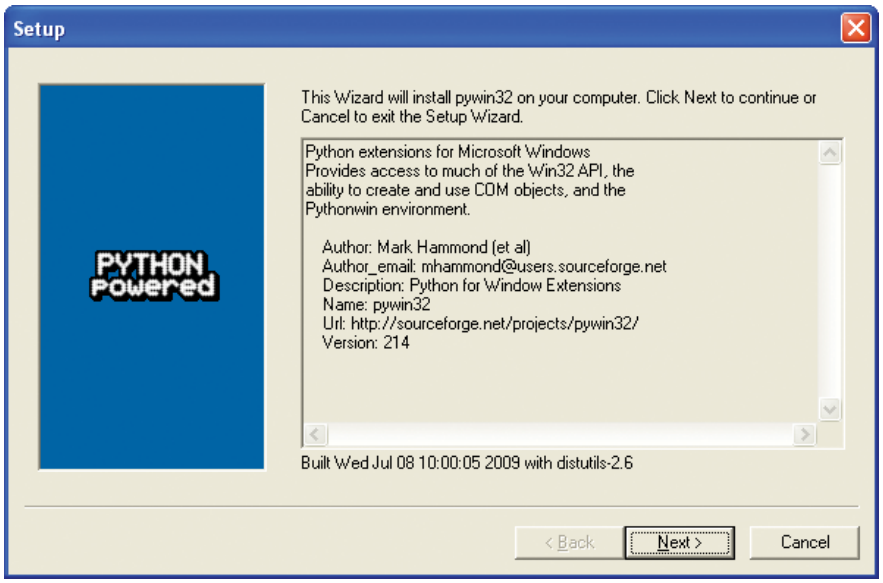
Now it is time to find the installation files for PythonWin.

4 Insert the disk containing the exercise data that comes with the book. Start Windows, and then click the Start button. Click My Computer and browse to the drive where the disk is installed. Browse to the C:\EsriPress\Python\Data\Exercise01 folder. This folder contains the file pywin32-214.win32-py2.7.exe.

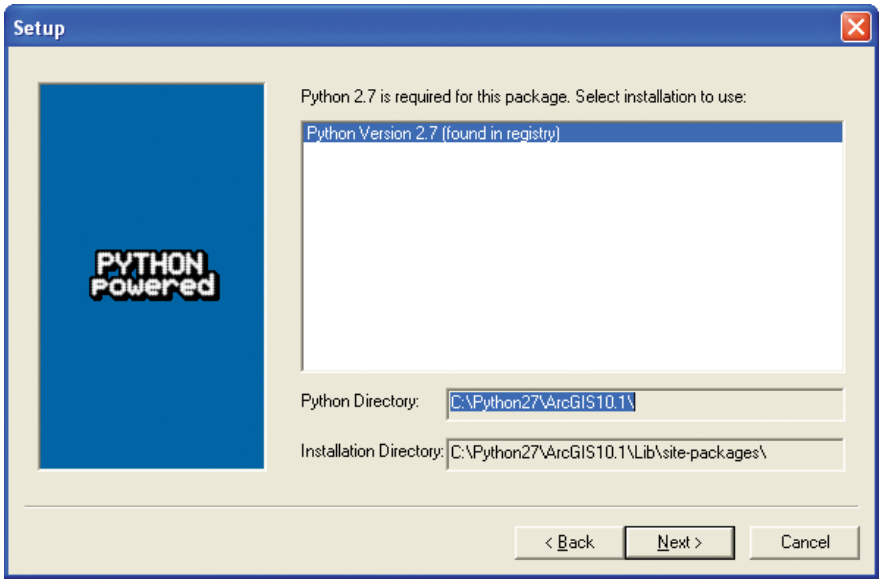
Once you have found the correct installation file, you can proceed with the installation.

5 Double-click pywin32-214.win32-py2.7.exe to launch the PythonWin installation.

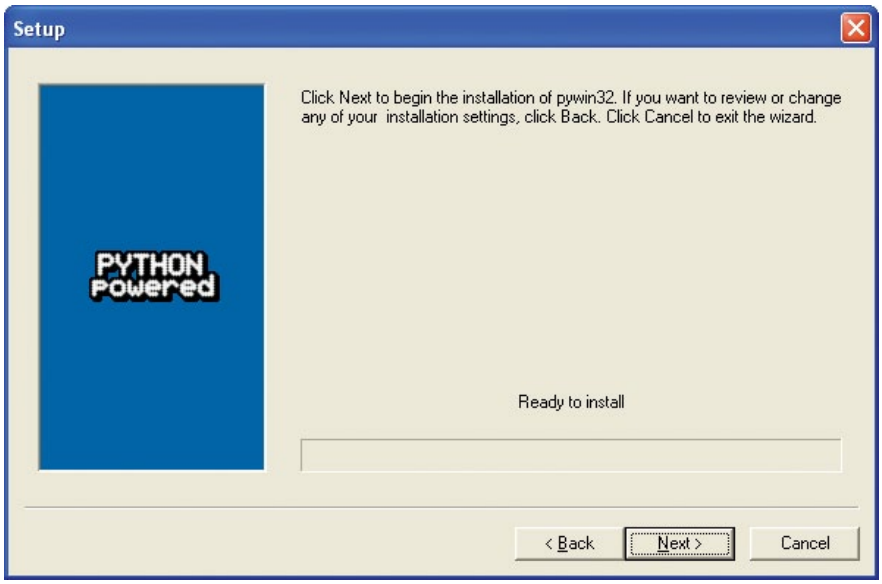
6 Follow the on-screen instructions. On the first panel of the Setup wizard, click Next.



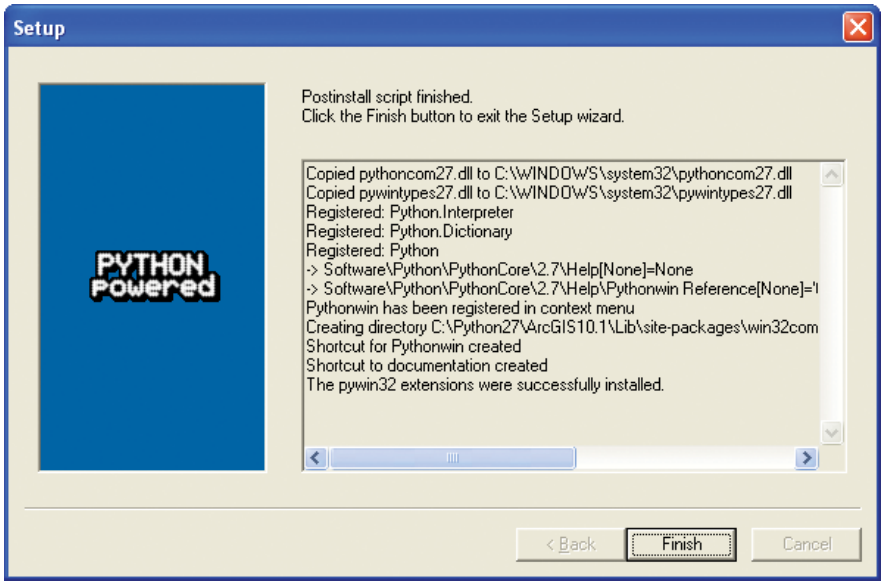
7 On the second panel of the Setup wizard, notice that the installation program found the Python version installed on your computer since it is installed with ArcGIS. Accept the default directories and click Next.



8 On the third panel of the Setup wizard, click Next.



9 On the final panel of the Setup wizard, click Finish.



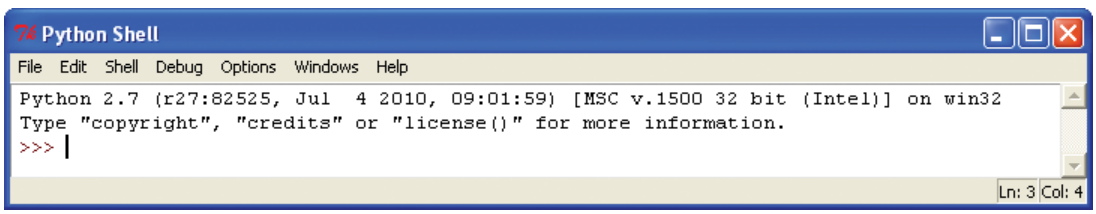
You are now ready to use PythonWin as your editor.

Note: *If at this last step you get an error message that a shortcut for PythonWin could not be created, retrace your steps and first create a program folder for Python 2.7.*

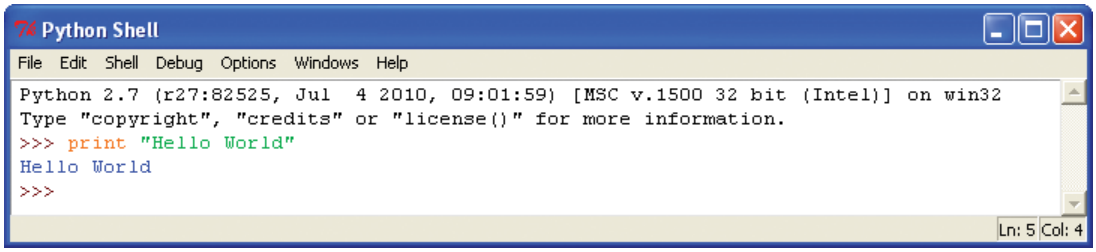
Work with different script editors

There are several ways to use Python to run scripts, but the most common is to use a script editor. The next set of steps shows how to use two of these script editors: IDLE and PythonWin.

1 On the taskbar, click the Start button, and then, on the Start menu, click All Programs > ArcGIS > Python 2.7 > IDLE (Python GUI). This launches the standard interactive interpreter, the Interactive Window.



2 Make sure the pointer is placed directly following the prompt (>>>). Type print "Hello World" **and press ENTER.**

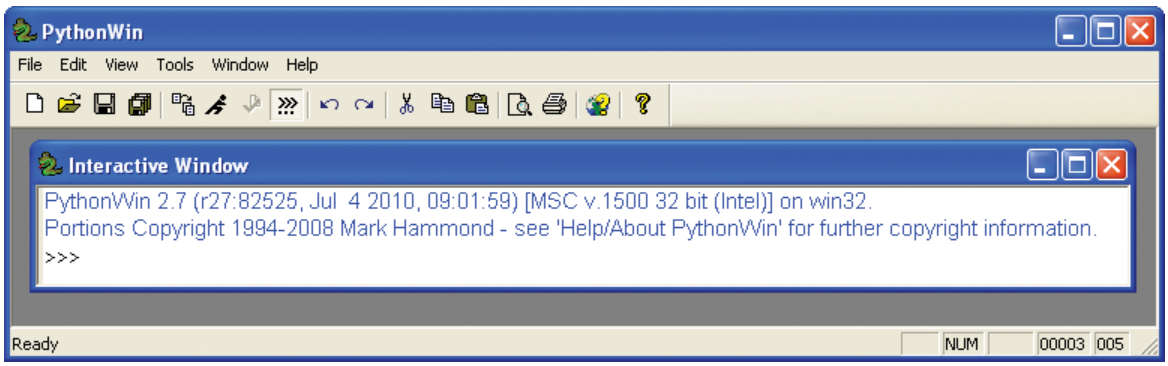


Notice that Python gives you the output of the line immediately. You just entered a Python statement. The print statement is used to print the value you supplied to the Interactive Window. The Interactive Window has a prompt with three angle brackets (>>>). When you type a line of code and press ENTER, that line is interpreted and run immediately. Notice that the interactive interpreter automatically recognizes statements (such as `print`) and values (such as `"Hello World"`) and shows them in a different color. This is called "syntax highlighting."

The IDLE interface is installed by default when Python 2.7 is installed. Although it is a solid editor for working with Python, for the exercises in this book you will use PythonWin because it has a number of advantages on the Microsoft Windows platform, in particular the integration of the code debugger.

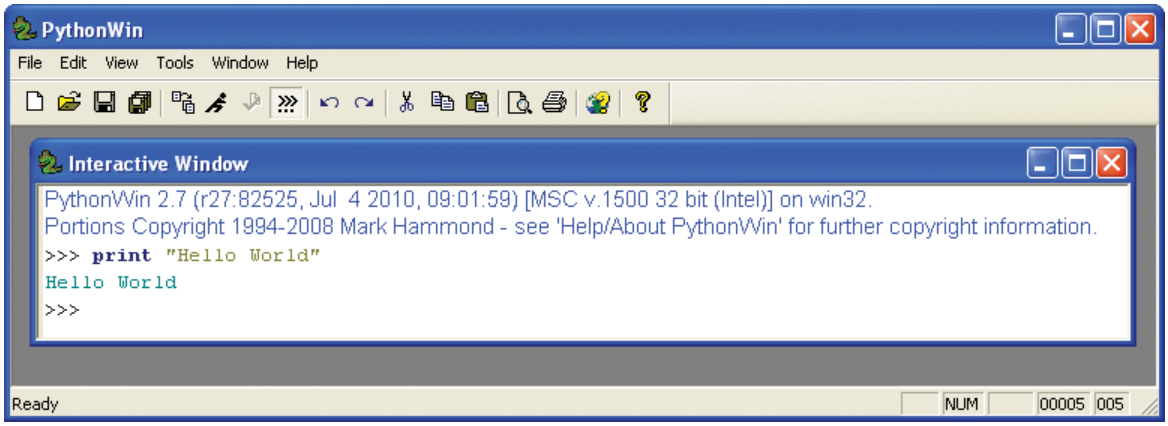
3 Close the IDLE editor by clicking File > Exit on the menu bar.

4 On the taskbar, click the Start button, and then, on the Start menu, click All Programs > Python 2.7 > PythonWin.



Now you can try the same `"Hello World"` statement.

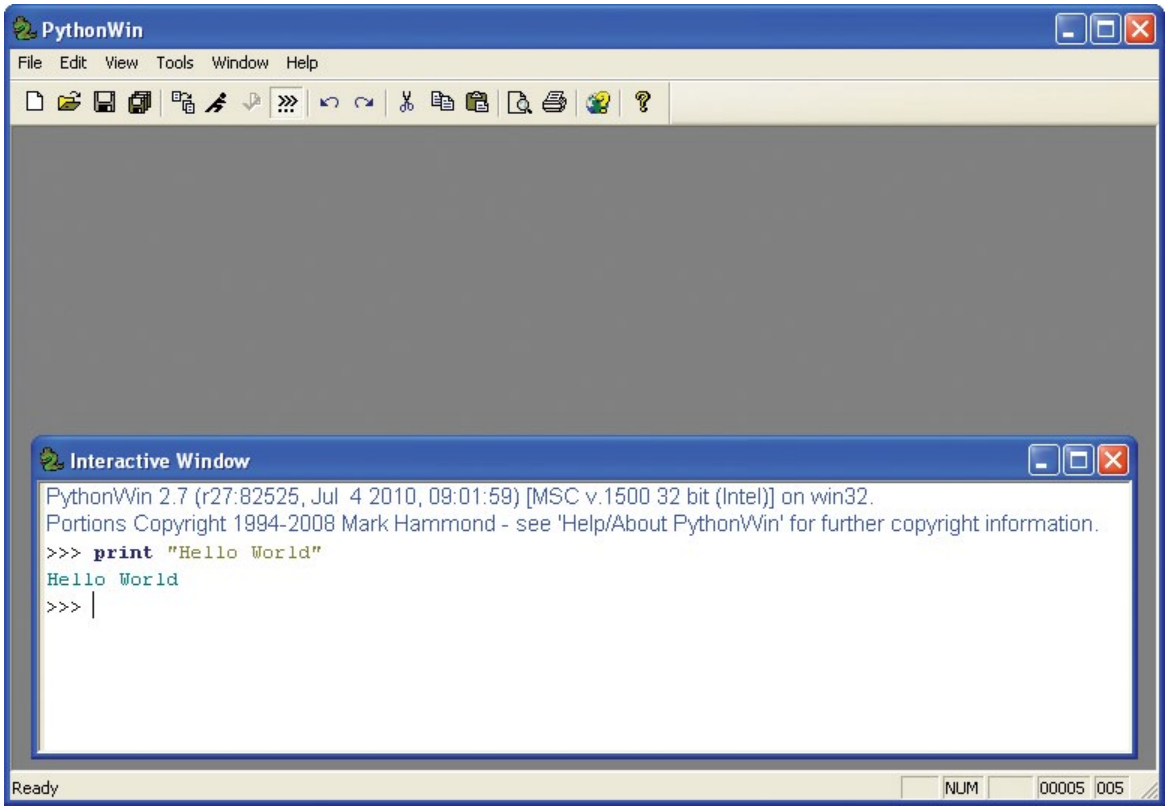
5 Make sure the pointer is placed directly following >>>. Type print "Hello World" **and press ENTER.**



Notice that PythonWin uses a slightly different style for syntax highlighting, but the concept is the same as in the IDLE editor. The instructions for the exercises in this book use PythonWin, but the syntax would be the same in another editor like IDLE.

When you launch PythonWin, the Interactive Window opens by default. You can quickly write and run code here, but the code itself is not saved. The Interactive Window also shows the outputs from print statements (as in the earlier example) and reports error messages from scripts. Typically, you keep the Interactive Window open, but you write scripts in a script window that allows you to save them.

6 Resize the Interactive Window so there is room for another window within PythonWin.



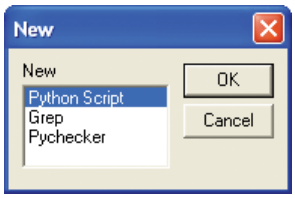
>>> TIP

You want the Interactive Window to be open to see any messages that may appear when you run a script. If you happen to close the Interactive Window, you can reopen it by going to the PythonWin menu bar and clicking View > Interactive Window. There is also a toggle button on the PythonWin Standard toolbar to hide or show the Interactive Window.

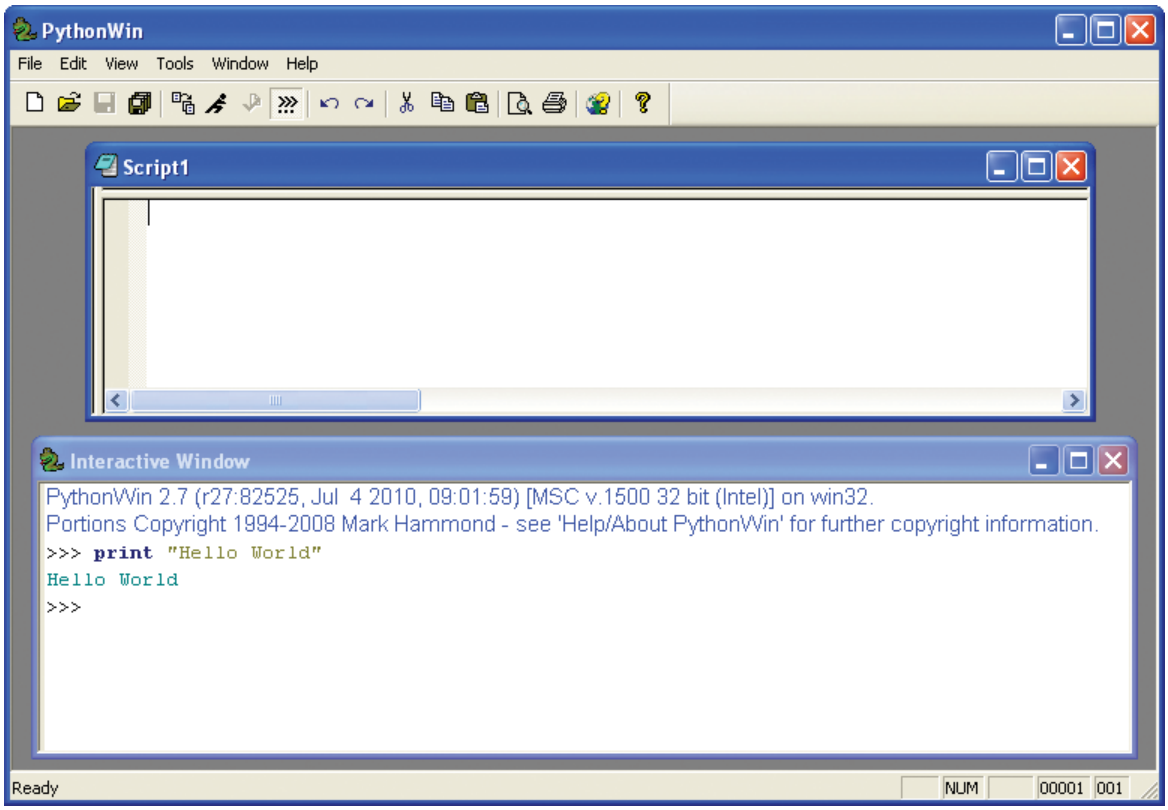
Now you can add a new window where you can write a script.

7 On the PythonWin menu bar, click File > New.

8 On the New dialog box that appears, click Python Script and click OK. →



You now have two windows open within PythonWin. Code written in the Script window can be saved, but code written in the Interactive Window cannot be. Outputs from the Script window are printed to the Interactive Window.



9 With the Script window active, on the PythonWin menu bar, click File > Save As. On the Save As dialog box, navigate to the folder C:\EsriPress\Python\Data\Exercise01 and save your file as hello.py.
The extension .py indicates the file is a Python script.

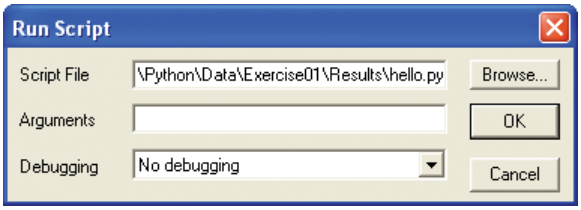
10 In the hello.py script window, type print "Hello World"**.**

Pressing ENTER, in this case, moves the cursor to the second line but does not actually run the code because this is a script, and not the Interactive Window. This allows you to keep writing code until you are ready to run the script.

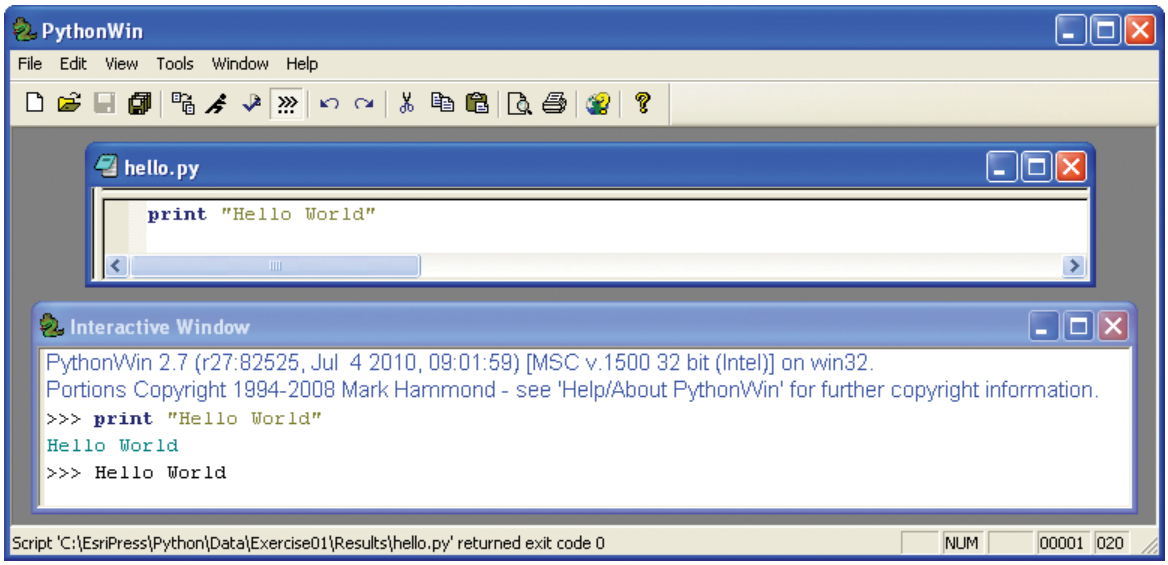
Note: There is no prompt in a script. Contrary to the Interactive Window, you don't run a script line by line, but in its entirety, as you will see in the steps that follow.

11 With the cursor still placed anywhere within the code in the hello.py script window, click Run . You can also click File > Run on the PythonWin menu bar or press CTRL+R.

12 This brings up the Run Script dialog box. For now, leave the default settings and click OK. →



This runs the script hello.py and prints the result to the Interactive Window.



Note: *Before you can run a script, it needs to be saved as a file with a .py extension. Once you save it, every time you run the script, it is saved again automatically. So remember that running a script automatically overwrites any earlier versions of the script that have the same name.*

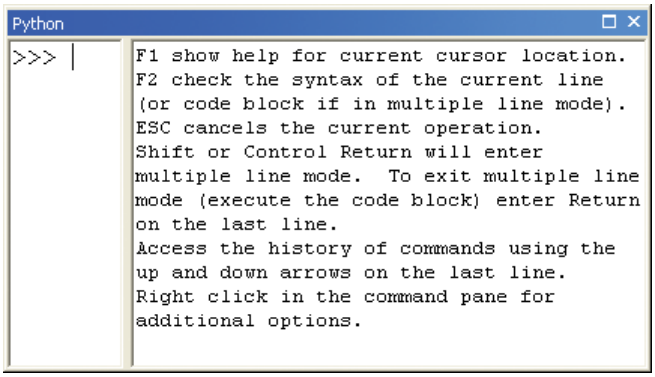
13 Close the hello.py script. You can do this by clicking the X symbol of the script window or by clicking File > Close on the PythonWin menu bar with the script window active.

14 Close PythonWin by clicking the X symbol of the PythonWin window or by clicking File > Exit on the menu bar.

Use the ArcGIS Python window

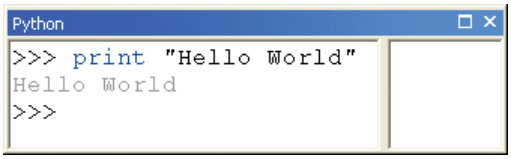
Both the IDLE and PythonWin script editors provide robust environments for working with Python, but ArcGIS 10 introduced another way to run Python code: the Python window. The Command Line from earlier versions was replaced in ArcGIS 10 with a fully interactive Python interpreter. Next, you can take a look at how it works.

1 Start ArcMap. On the Standard toolbar, click the Python window button . This opens the Python window. On the left side is the Python interpreter, and on the right is the Help and syntax panel. Notice that you can use the F1 and F2 keys to access more help. The divider between the two sides can be moved by dragging it. ➔



By default, the cursor is placed on the first line, just after the prompt (>>>). Visually, it may appear as if there is a blank space between the prompt and the cursor, but there is actually no space to delete.

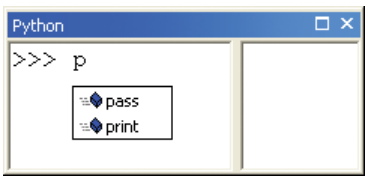
2 Following the prompt, type print "Hello World" **and press ENTER.** Notice that the Python window works very similarly to the Interactive Window in PythonWin and IDLE. ➔



There are a number of useful features when using the Python window.

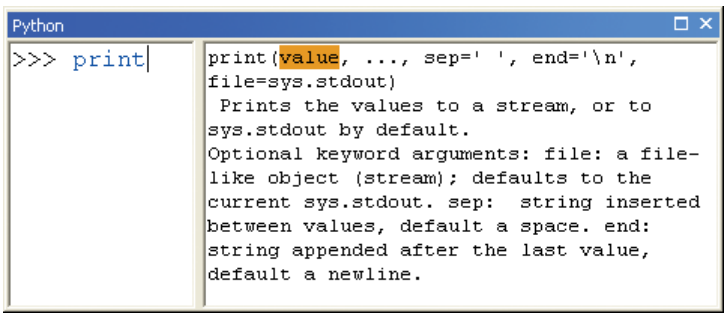
3 Right-click next to the prompt (>>>) and click Clear All.

4 Start typing the letter *p*. Notice that this results in the suggestions shown in the figure. ➔



Only two statements in Python start with the letter *p*, and these are provided as a list of autocompletion prompts. You can continue typing your statement or select from the prompts.

5 To select from the prompts, use the UP ARROW and DOWN ARROW keys to navigate or click a prompt to highlight it, and then use the TAB key to enter the prompt into the line of code. ➔



The use of prompts is a fast way to write code and prevent typos. It is also an effective way to learn Python commands, because it provides you with a list of the possible choices.

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

ArcGIS Desktop Help lists several key features that make the Python window a valuable resource for running and experimenting with Python commands and syntax. In ArcGIS 10.1 Desktop Help, on the Contents tab, search for "Using the Python window" and you'll find the following:

- All Python functionality is exposed through the Python window.
- Multiline commands that contain more than one geoprocessing tool or geoprocessor method can be entered and executed.
- Tools or functions that have already been entered and executed can be recalled, edited, and re-executed.
- Python commands or blocks of code can be loaded from existing Python files.
- Python commands or blocks of code can be saved to a Python or text file to be reloaded later or used in a different environment.
- Autocompletion functionality makes filling in geoprocessing tool parameters quicker and easier than using tool dialog boxes.

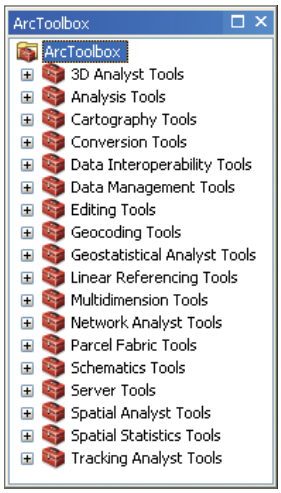
Several of these features of the Python window will be explored in later exercises.

Exercise 2
Geoprocessing in ArcGIS

Examine toolboxes and tools

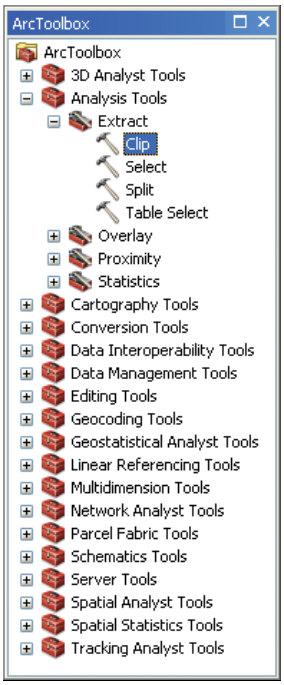
ArcGIS software contains hundreds of tools, organized in toolboxes and toolsets. There are two main ways to find the tools you need: search and browse. You will practice both.

1 **Start ArcMap. On the Standard toolbar, click the ArcToolbox button . You can also open the ArcToolbox window from the menu bar by clicking Geoprocessing > ArcToolbox.**

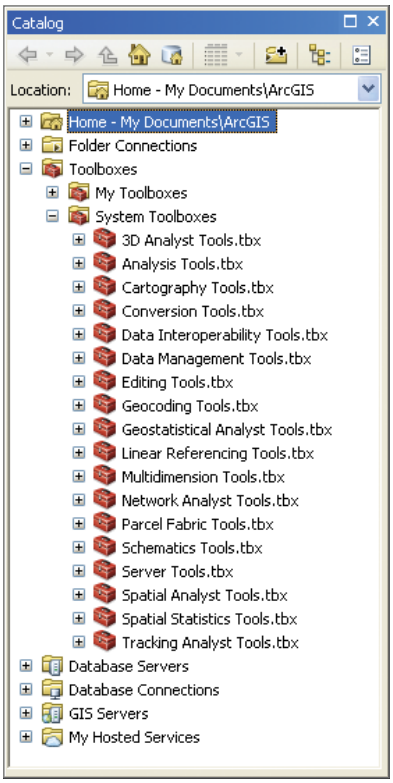


The ArcToolbox window shows a list of all the system toolboxes in ArcGIS and allows you to browse through an organized list of all the tools.

2 **Expand the Analysis toolbox and then the Extract toolset to see the list of tools inside this toolset.** ➔



Finding the tool you want can be a bit cumbersome if you are not sure where to look, but with repetition you will start to remember where the tools you use most are located. A similar approach to browsing can be accomplished in the Catalog window.



3 **Close the ArcToolbox window.**

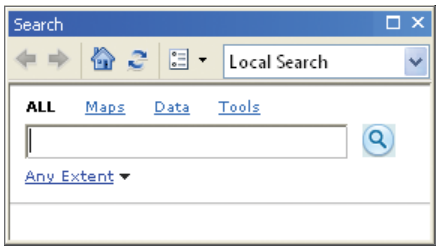
4 **On the Standard toolbar, click the Catalog button .**

5 **Expand the Toolboxes entry, and you can see two folders: one for your custom toolboxes (My Toolboxes) and one for system toolboxes. The organization of the system toolboxes is the same as in the ArcToolbox window.** ➔

Now you can try using the Search window to find the geoprocessing tools.

6 **Close the Catalog window.**

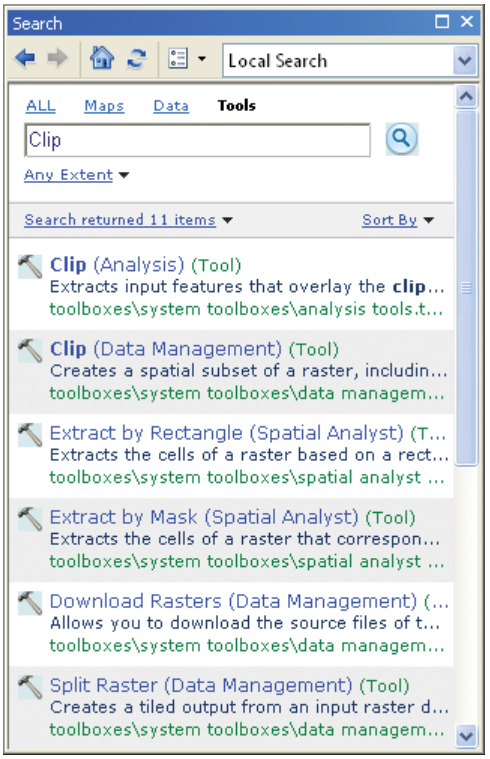
7 **On the Standard toolbar, click the Search window button . You can also open the Search window from the menu bar by clicking Geoprocessing > Search For Tools.**



8 **In the Search window, click the Tools hyperlink and increase the size of the Search window to see the complete list of toolboxes.** These are exactly the same toolboxes you've seen in the ArcToolbox and Catalog windows. So you can browse to the tools you need in a similar fashion. ➔

In addition to browsing, you can also search for tools by name.

9 Place your pointer in the Search box, type clip**, and click the Search button** . →



The results panel shows the tools that have the search term "clip" in the name or in the description. In this case, notice that there is a Clip tool for vector data such as polylines and polygons in the Analysis toolbox and a Clip tool for raster data in the Data Management toolbox.

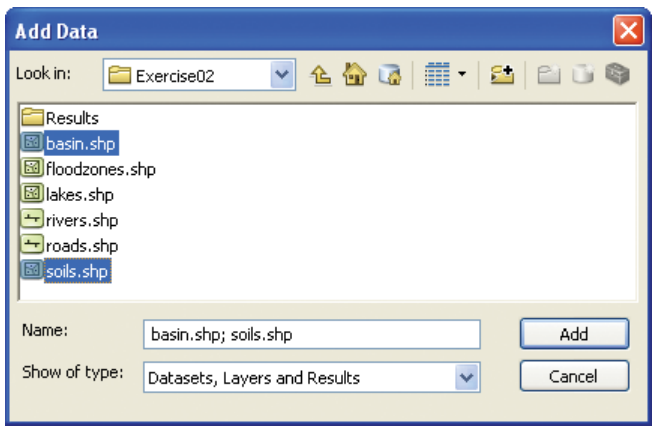
You can click the link to the Item description to get a more complete description of how the tools work—this is the same material as provided in the full ArcGIS Desktop Help files.

Following the link to the path, toolboxes\system toolboxes, opens the Catalog window. Double-clicking the tool name opens the tool dialog box.

10 Close the Search window.

Run a tool

Next, you will add some data so you can start using some of the geoprocessing tools.

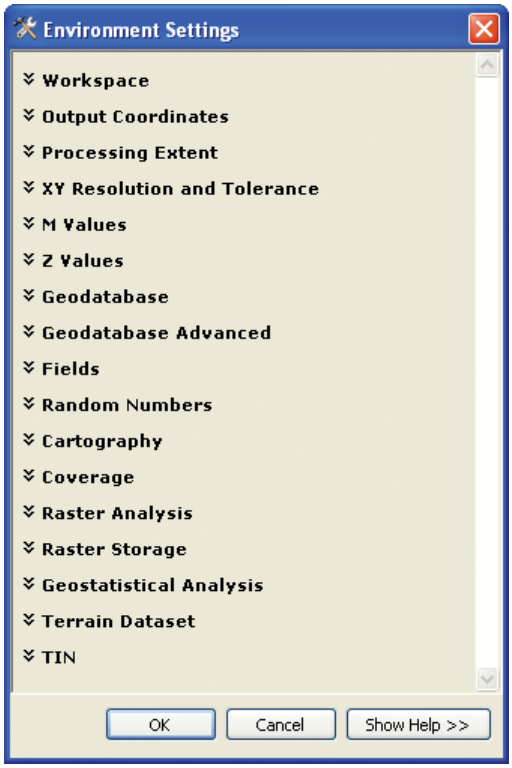


1 With ArcMap open, on the Standard toolbar, click the Add Data button . **On the Add Data dialog box, browse to the folder C:\EsriPress\Python\Data\Exercise02. Hold down CTRL and click basin.shp and soils.shp. Click Add.** →

This adds two shapefiles to your current data frame in ArcMap. Check to see that the extent of the soils shapefile is much larger than the basin shapefile. You will use the Clip tool to reduce the extent of the soils shapefile to match the basin shapefile.

Prior to running geoprocessing tools, you should set your environments. Environments are parameters that control how a tool is run, but these settings do not appear on the tool dialog box.

2 On the menu bar, click Geoprocessing > Environments. This brings up the Environment Settings dialog box. ➔

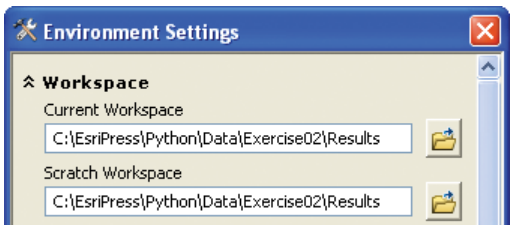


Notice the large number of categories, each with a number of options under it. In this case, you will set only the Workspace.

3 Click the Workspace entry to expand its options.

4 Click the Browse button next to Current Workspace. Navigate to the C:\EsriPress\Python\Data\Exercise02 folder and click Results. Click Add.

5 Repeat step 4 to set the Scratch Workspace.



By setting the workspace, the outputs of geoprocessing operations will now be saved to the C:\EsriPress\Python\Data\Exercise02\Results folder. Instead of specifying a folder, you can also specify an existing geodatabase. In this case, the current workspace and the scratch workspace are set to the same folder, but they can be set to different ones—this can be useful for separating intermediate results from final results in geoprocessing workflows that have many different steps generating dozens to hundreds of outputs. This is typically the case when using models in ModelBuilder.

6 Click OK to close the Environment Settings dialog box.

There are many other environment settings, some of which are introduced in later exercises.

Now you are ready to run a tool.

7 Open the ArcToolbox window. Expand the Analysis toolbox and then the Extract toolset and double-click the Clip tool.

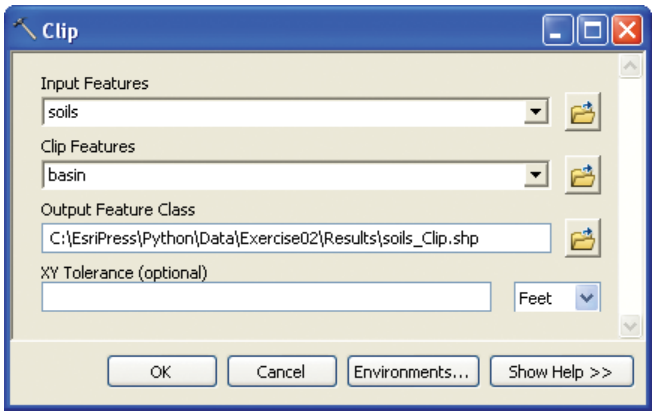
Next, you will specify the parameters of the Clip tool.

Note: *These instructions primarily use the ArcToolbox window to find and run tools, but you can also use the Catalog or Search windows.*

8 **For Input Features, select the soils shapefile from the drop-down list.**

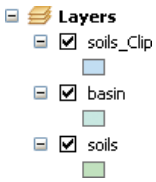
9 **For Clip Features, select the basin shapefile from the drop-down list.** Notice that the Output Feature Class is automatically populated. The path is determined by the environment settings (C:\EsriPress\Python\Data\Exercise02\Results) and the file name is based on the inputs and the name of the tool. You can use the Browse button to navigate to a different path. You can also change the path and output file name by typing in the text box.

10 **Leave the XY Tolerance blank.**



11 **Click OK to run the tool.** A progress bar that appears at the bottom of the document on the ArcGIS for Desktop application status bar shows that the tool is running. Once tool execution is complete, a small pop-up notification briefly appears in the notification area, at the far right of the taskbar.

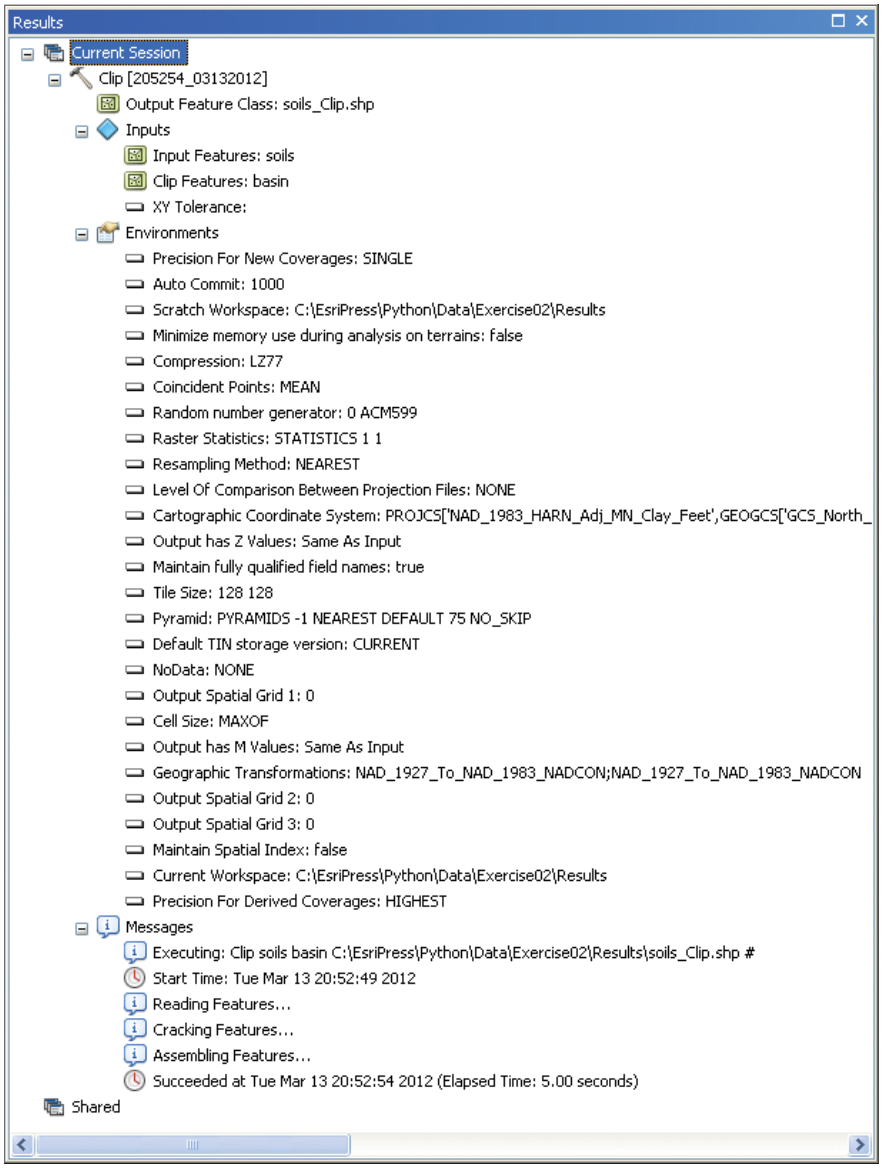
The result of running the tool is a new shapefile called **soils_Clip**, which is added to the ArcMap table of contents.



You can now explore the soils_Clip shapefile to confirm the result. You can also review the execution of the tool by exploring the Results window.

12 **On the ArcMap menu bar, click Geoprocessing > Results.**

13 In the Results window, expand the Current Session entry as well as the entries inside Current Session.



The Results window records a history of the geoprocessing operations, including all the tool parameters and the environment settings. The messages at the end demonstrate the steps that were part of running the tool and show when the run was completed.

14 Close the Results window.

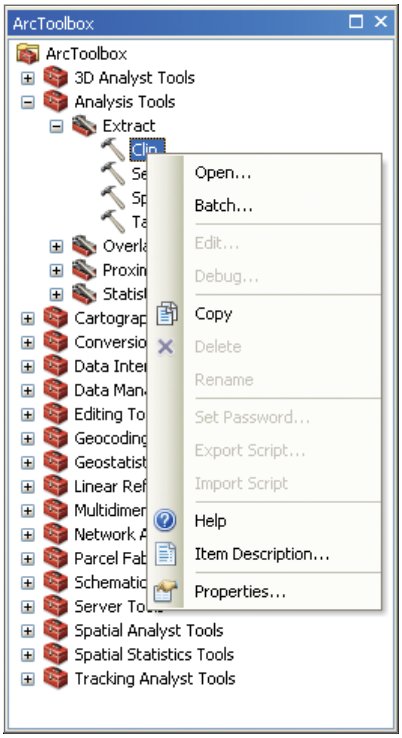
15 Right-click the soils_Clip layer in the ArcMap table of contents and click Remove.

Conduct batch processing

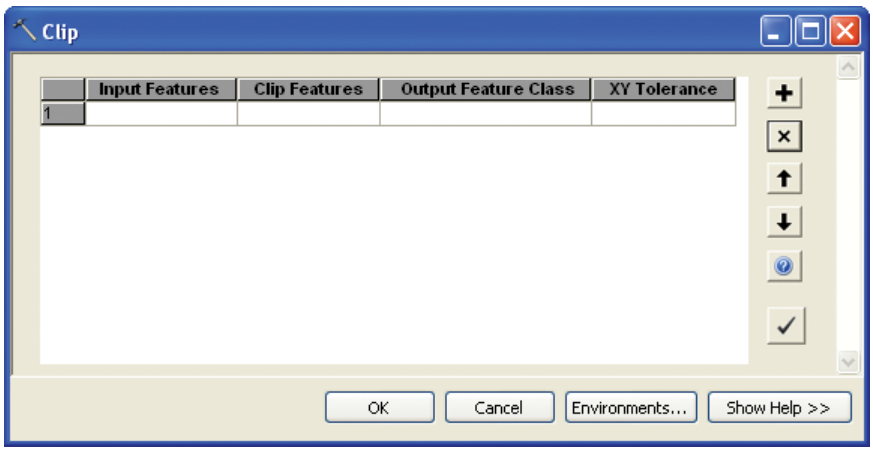
You will next use batch processing to clip the other layers.

1 Add all the other layers in the exercise 2 folder to the data frame: floodzones, lakes, rivers, and roads.

2 Right-click the Clip tool and click Batch.



This brings up the batch grid for the Clip tool.

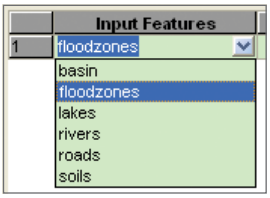


There are several ways to fill in the cells, as follows:

- Double-clicking a row number brings up the Clip tool dialog box for that row.
- Double-clicking in a cell brings up a dialog box for just that cell.
- Clicking on the right side of a cell opens a drop-down list of layers.
- Right-clicking in a cell and clicking Open brings up a dialog box for that cell.
- Right-clicking in a cell and clicking Browse brings up a dialog box for browsing.

Since the layers have already been added to the data frame, using the drop-down option in this case is quite efficient.

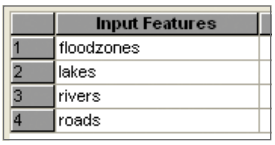
3 In the column Input Features, click on the right side of the cell in the first row to open the drop-down list and select the floodzones layer. ➔



4 Add a row to the batch grid by clicking the Add Row button + **.**

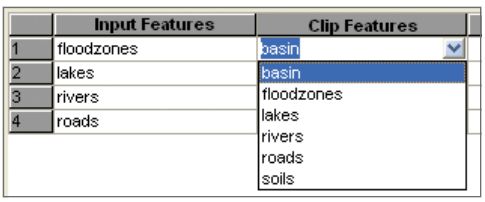
5 In the second row, use the drop-down arrow to select the lakes layer.

6 Keep adding rows for the remaining layers that need to be clipped so that the grid matches the example in the figure. ➔

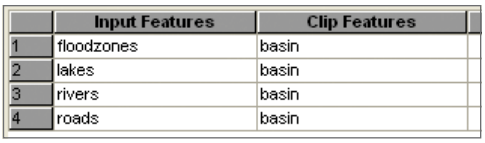


Now you can move on to the Clip Features column.

7 For Clip Features in the first row, use the drop-down list to select the basin layer. ➔

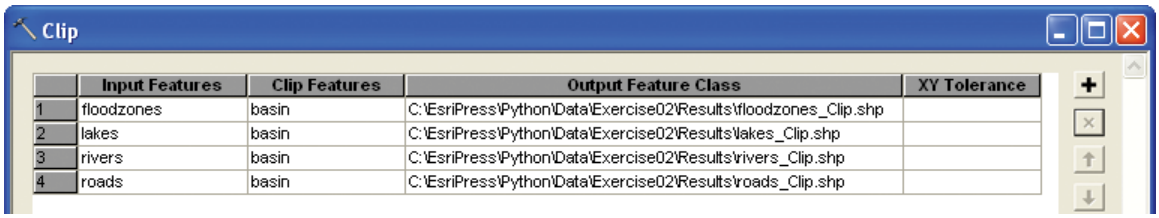


8 In the first Clip Features cell, right-click basin and click Fill. It populates the cells in the remaining rows with the same value—that is, the basin layer. ➔



The last set of parameters to fill in are for Output Feature Class.

9 Click the Check Values button. It validates the cell values in the batch grid, and in the case of Output Feature Class, populates the file names with defaults based on the current workspace.



You can also enter the file names one by one, but for filling in a large number of rows, the Check Values button provides a quick method. You can make changes to these values by modifying a single cell (for example, by double-clicking in the cell or right-clicking in the cell and clicking Open).

The batch tool is now ready to run.

10 Click OK. Once the tool runs, the four output feature classes are added to the ArcMap table of contents.

Set the geoprocessing options

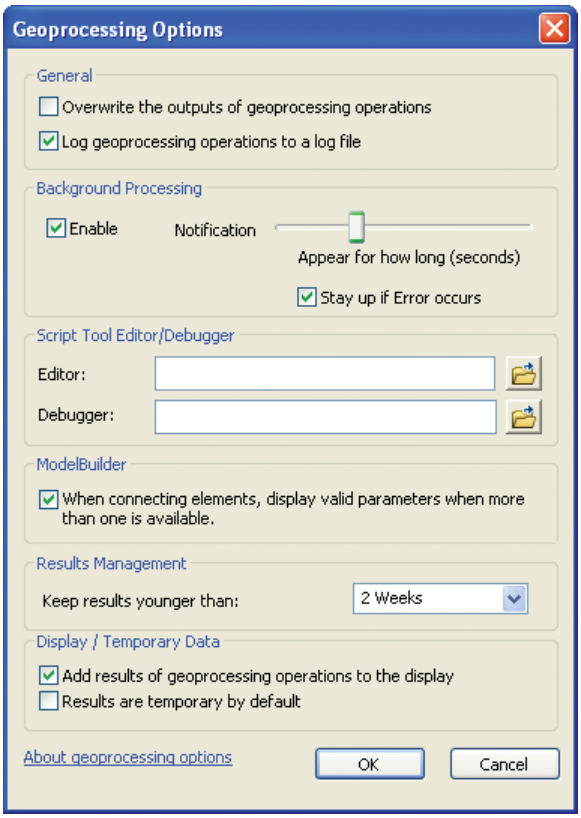
Sometimes when you run geoprocessing operations, you encounter a situation where you might want to overwrite the existing data. For example, you may realize you made a mistake and want to run a tool again using the same name you already used for an output feature class. This is a common scenario when running models. By default, geoprocessing tools do not overwrite existing datasets, but this is a setting you can change.

First, you will modify the geoprocessing option to prevent overwriting files.

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

2 Under the General heading, clear the "Overwrite the outputs of geoprocessing operations" check box. ➔

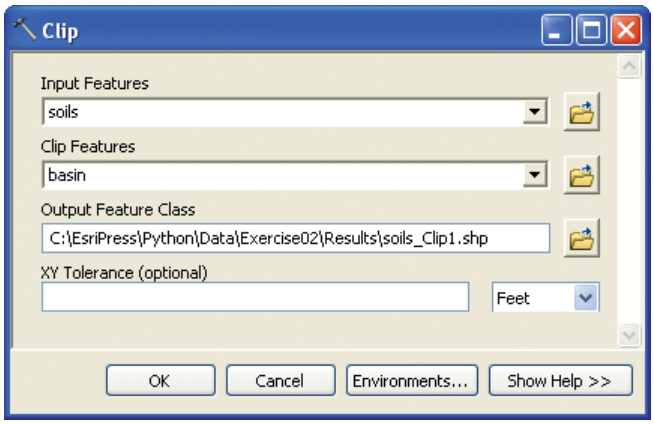
3 Click OK.



Next, take a look at the following example.

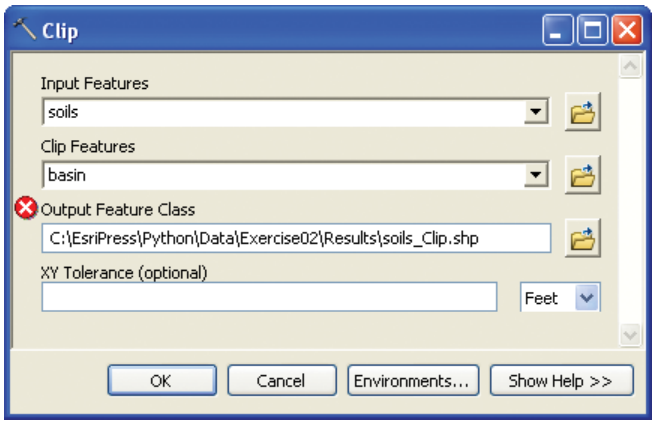
4 Open the Clip tool dialog box.

5 For Input Features, select soils and for Clip Features, select basin.



Notice that, by default, in the naming of the output feature class, ArcMap recognizes that the soils_Clip feature class already exists and therefore the name soils_Clip1 is entered.

6 Change the name of Output Feature Class to soils_Clip**. Then click outside the input box for this parameter.** Since this name already exists, an error icon appears indicating that the tool will not run.

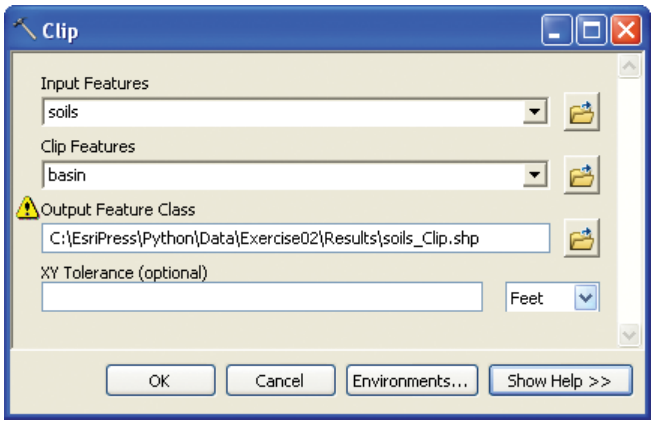


Next, modify the geoprocessing option to allow for overwriting files.

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

8 Under the General heading, select the "Overwrite the outputs of geoprocessing operations" check box.

9 **Click OK.** On the Clip tool dialog box, the error icon has become a warning icon. The warning message says that the output feature class soils_Clip already exists, but this will not prevent the tool from running.



10 **Click Cancel to close the Clip tool.**

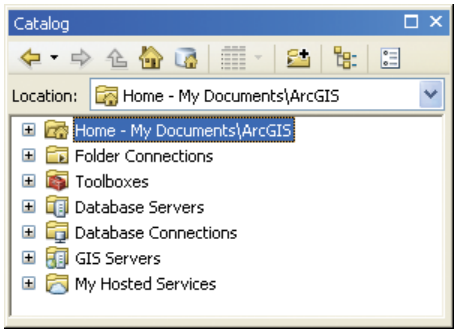
>>> TIP

Although the changes to the geoprocessing options take effect immediately, you may need to retype the name of the output feature class on the Clip tool dialog box for the error icon to disappear.

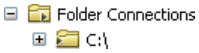
Explore models and ModelBuilder

Models are one way to create a sequence of geoprocessing operations in ArcGIS. Like tools, models are organized in toolboxes and toolsets within ArcToolbox. Before creating a model then, you need to create a toolbox to store it.

1 **On the ArcMap Standard toolbar, open the Catalog window by clicking the Catalog button.**



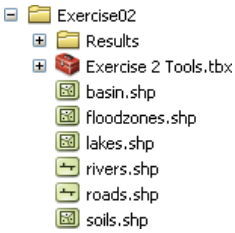
2 **Expand Folder Connections and make a connection to the C drive if this connection does not already exist.**



3 **Navigate to the exercise 2 folder.**

4 Right-click this folder and click New > Toolbox.

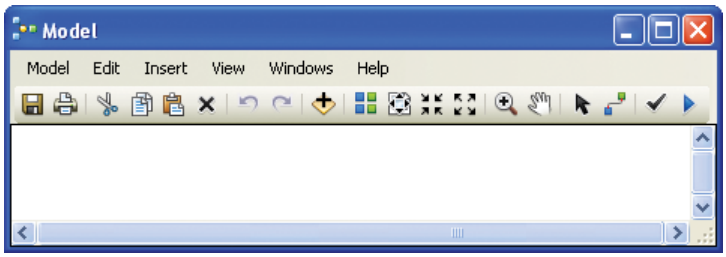
5 Name the toolbox Exercise 2 Tools.tbx**.**



Note: *You can create your toolbox in any folder or in a geodatabase. The key is that a model needs to be saved within a toolbox, so you must create a toolbox first before creating the model.*

Now you are ready to create a model.

6 In the Catalog window, right-click the Exercise 2 Tools toolbox and click New > Model. This brings up the ModelBuilder interface and a new blank model.



First, you can give the model a name.

7 On the ModelBuilder menu bar, click Model > Model Properties.

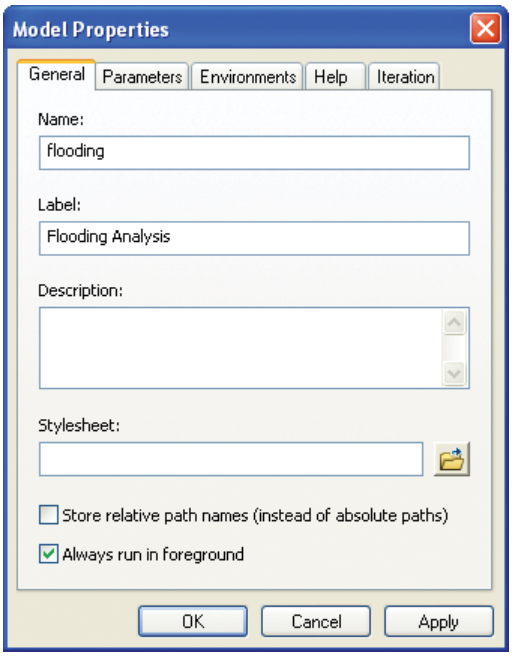
8 For Name, type flooding**. For Label, type** Flooding Analysis**.** ➔

Note: *The name indicates the actual name of the model. This name is used when calling the model from other tools in ArcGIS. The name cannot contain any spaces. The label indicates the label that will appear next to the model tool in the toolbox. The label can contain spaces.*

9 On the General tab, select the "Store relative path names" check box.

10 Click OK to close the Model Properties dialog box.

11 On the ModelBuilder toolbar, click the Save button.

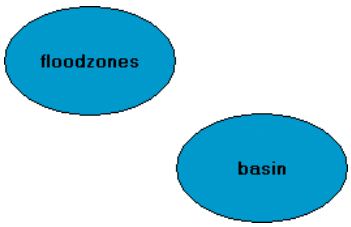


Notice that this updates the label of the model in the Exercise02 folder and the heading of the model itself. You are now ready to start adding elements to the model. There are several ways to add data and tools to your model, as follows:

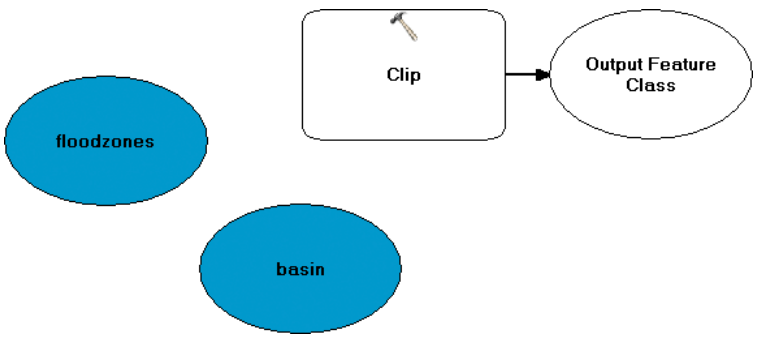
- Use the Add Data or Tool button on the ModelBuilder toolbar.
- Right-click in the model and click Add Data or Tool.
- Drag layers from the table of contents and tools from ArcToolbox into the model.

You will use the drag-and-drop method in the following steps, but the other methods are just as good.

12 Drag the basin and floodzones layers from the ArcMap table of contents into the model.



13 Drag the Clip tool from ArcToolbox into the model.



As the layers were added, their oval symbols were given a fill color (blue) because the file name for these data variables is specified. When the Clip tool was added, its rectangular symbol remained hollow because the tool's parameters have not been specified yet. Hollow symbols indicate that a model is not ready to run. In addition, by its very nature, the Clip tool produces an Output Feature Class, so this data variable is automatically added to the model, even though it is not pointing to an output feature class yet.

Connectors also need to be added to make the model ready to run. In this example, the layers basin and floodzones need to be connected to the Clip tool. There are two ways to create the appropriate connectors:

1. Use the Connect tool . On the toolbar, click this tool, and then click one element and drag the connector to the second element.

2. Open the tool dialog box and specify the tool's parameters.

In the following steps, you will use the tool dialog box option.

14 In the model (not in ArcToolbox!), double-click the Clip tool to open the tool dialog box.

You can now specify the tool's parameters as you normally would for a tool.

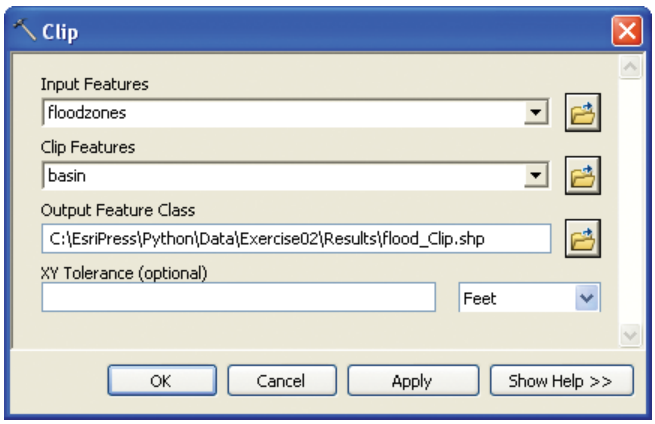
15 For Input Features, click the drop-down arrow.

The drop-down list shows all the available layers in the ArcMap table of contents as well as the data variables already added to the model. This explains why floodzones and basin occur twice on the list. →

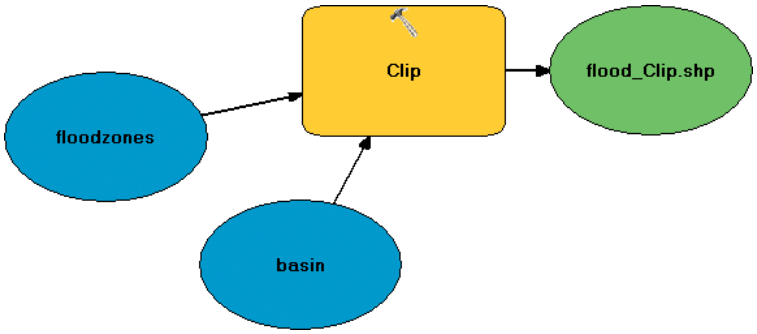
16 Click the floodzones data variable (the one with a blue symbol in front of it) for Input Features.

17 Click the basin data variable for Clip Features.

18 Set Output Feature Class to C:\EsriPress\Python\Data\Exercise02\Results\flood_Clip.shp**.**

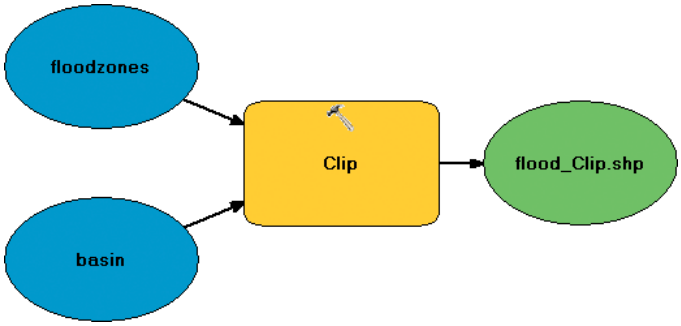


19 Click OK to close the tool dialog box. When you click OK to close the tool dialog box, the tool does not run as it would if you were using the tool in stand-alone mode—that is, outside a model. Instead, with the tool parameters specified, the appropriate connectors are created in the model. As a result, the symbol for the Clip tool in the model is now given a fill color (yellow) and so is the symbol for the output data variable (green). When all parameters are specified and all elements in the model have a fill color, the model is ready to run.



Before proceeding, try cleaning up the look of the model a bit.

20 On the ModelBuilder toolbar, first click the Auto Layout button and then the Full Extent button . This organizes the model elements into a consistent pattern. Although it has no effect on running the model, it makes it easier to follow the workflow in your model. As a general rule, after every few modifications or additions to a model, it is a good idea to use the Auto Layout and Full Extent buttons to reorganize the model elements.



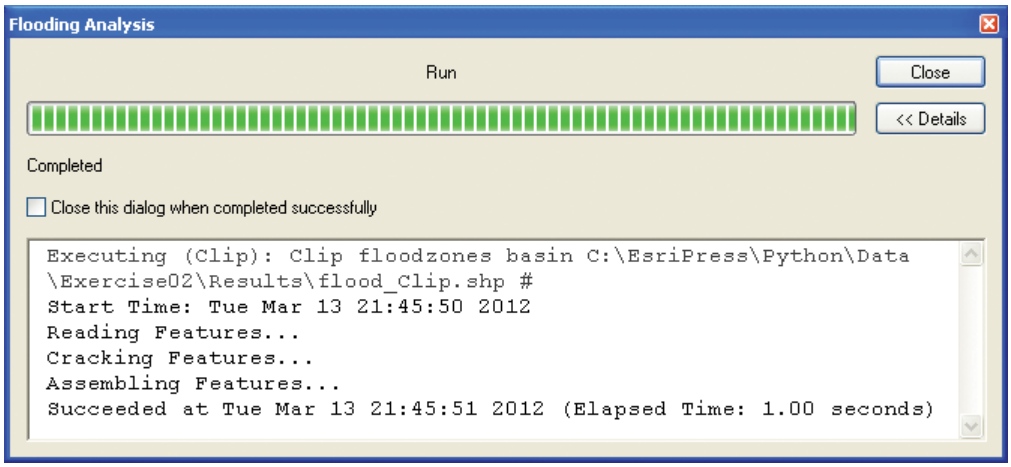
Run your model

Now your model is ready to run. There are several ways to run a model:

- You can click the Run button on the ModelBuilder toolbar to run the entire model.
- You can click Model > Run on the ModelBuilder menu bar to run the entire model.
- You can right-click a particular tool in the model and click Run to run just the selected tool.

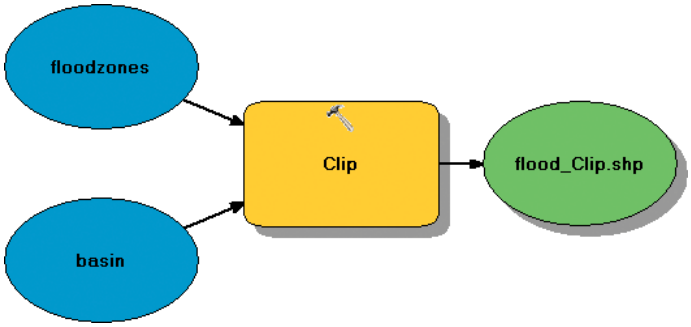
Since there is only one tool in the current model, there is no difference between running the entire model or only a single tool, but this option becomes more relevant when your models become more complex.

1 **On the ModelBuilder toolbar, click the Run button to run the entire model.** A model progress dialog box appears that shows the progress and time elapsed in running the tools in the model. The messages are similar to those in the Results window when geoprocessing tools are running.



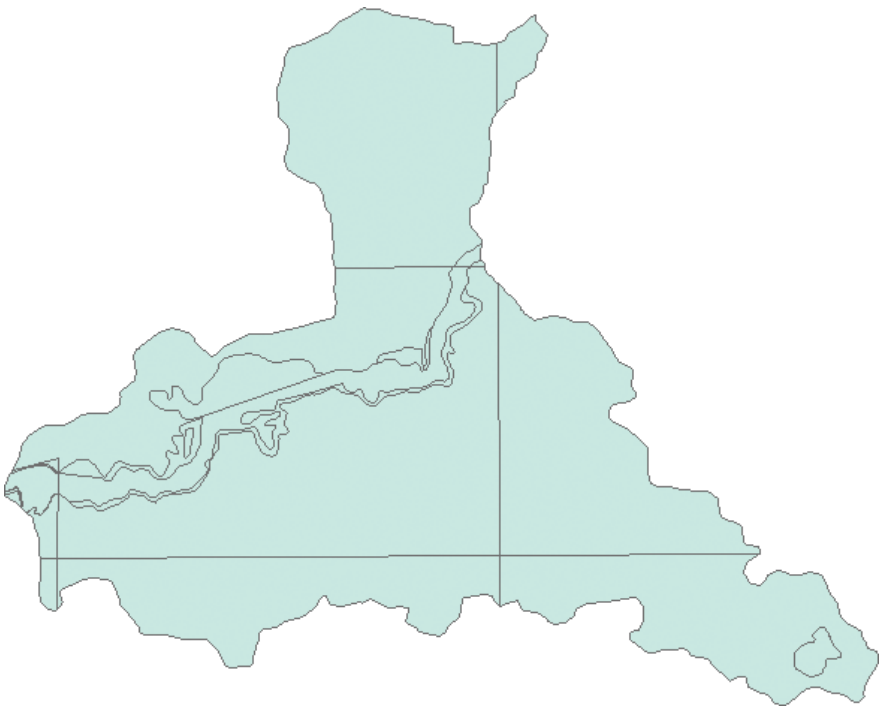
Note: *Since you are running the model from within ModelBuilder, the execution of the model is not recorded in the geoprocessing Results window, but on the model progress dialog box. If you were to save your model and close it, you could run it as a tool, and then the tool execution would be recorded in the Results window.*

2 Click Close to close the Flooding Analysis dialog box. When the model run is completed, the model elements (other than the input datasets) have a drop shadow to indicate that the tool has been run and the output datasets have been created—in this case, a shapefile.



Although the output shapefile flood_Clip.shp was created, it has not been added to the ArcMap table of contents. By default, ModelBuilder assumes the model outputs represent intermediate data.

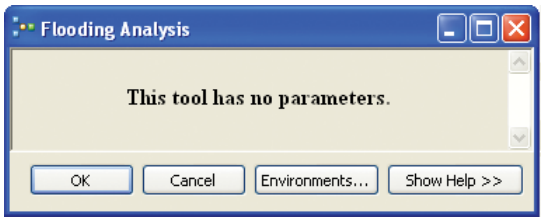
3 Right-click the flood_Clip.shp element in the model and click Add To Display. Then right-click the flood_Clip layer and click Zoom To Layer. You can now confirm that the flood_Clip layer has been added to the ArcMap table of contents and that it represents the clipped version of the floodzones layer.



4 On the ModelBuilder toolbar, click the Save button. Then on the menu bar, click Model > Close.

Next, you will run the model.

5 In the Exercise 2 Tools toolbox, double-click the Flooding Analysis model. This brings up the Flooding Analysis tool dialog box with the rather discouraging message, "This tool has no parameters." So what happened? Where is the model?



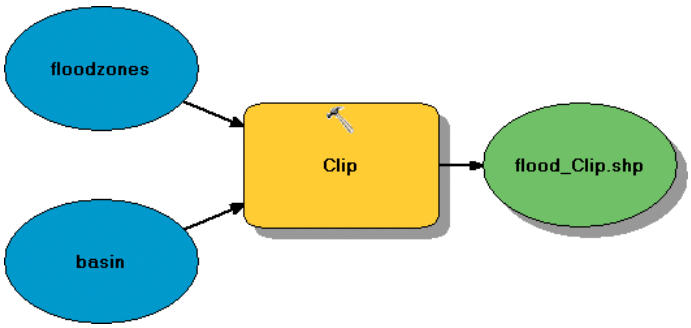
Models are tools, so by creating a model, you automatically create a tool. And tools have tool dialog boxes to specify parameters. However, in the ModelBuilder interface, you only created the model elements without indicating which elements should become parameters. In other words, the model is not yet fully ready to be used as a tool in which the user could specify the tool parameters.

Note: Creating tool parameters is not covered here but is covered in chapter 13 on custom tools.

So instead of running the model as a tool, you are going to go back into the model itself.

6 Click Cancel to close the Flooding Analysis tool dialog box.

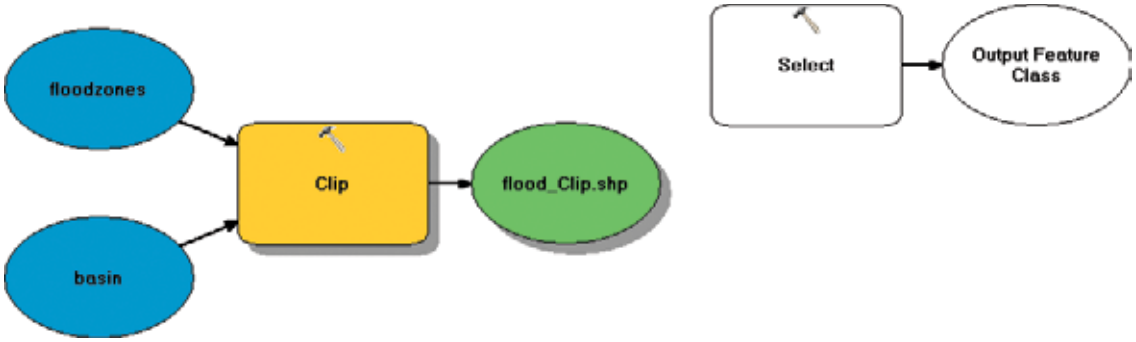
7 In the Exercise 2 Tools toolbox, right-click the Flooding Analysis tool and click Edit. This brings you back into the ModelBuilder interface. Notice that the Clip tool and the output feature class still have drop shadows—the model has already been run and it remembers its processing state.



Next, you can add another step to the model. You may have noticed that the polygons in the floodzones layer cover the entire study area. This is how traditional flood maps are organized: polygons cover the entire

study area but are coded as being inside or outside particular flood zone categories. You will add a tool to select just the polygons of interest.

8 In ArcToolbox, drag the Select tool from the Extract toolset in the Analysis toolbox into the model.



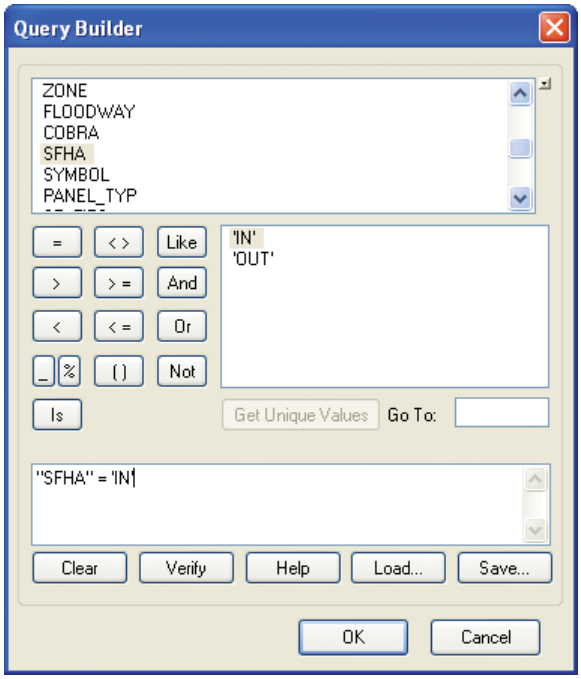
9 In the model (not in ArcToolbox), double-click the Select tool.

10 For Input Features, select flood_Clip.shp from the drop-down list.

11 Set the Output Feature Class to C:\EsriPress\Python\Data\Exercise02\Results\flooding.shp**.**

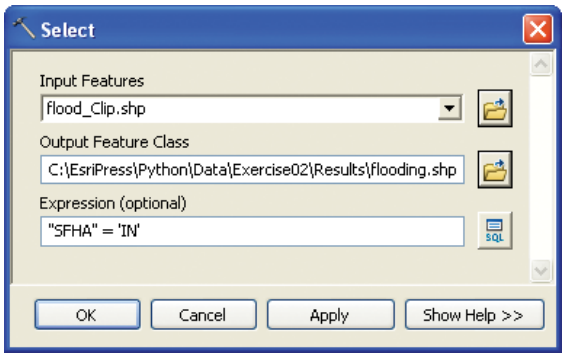
12 To create the Expression, click the SQL button .

13 On the Query Builder dialog box, create the following expression: `"SFHA" = 'IN'` (SFHA stands for Special Flood Hazard Area). ➔

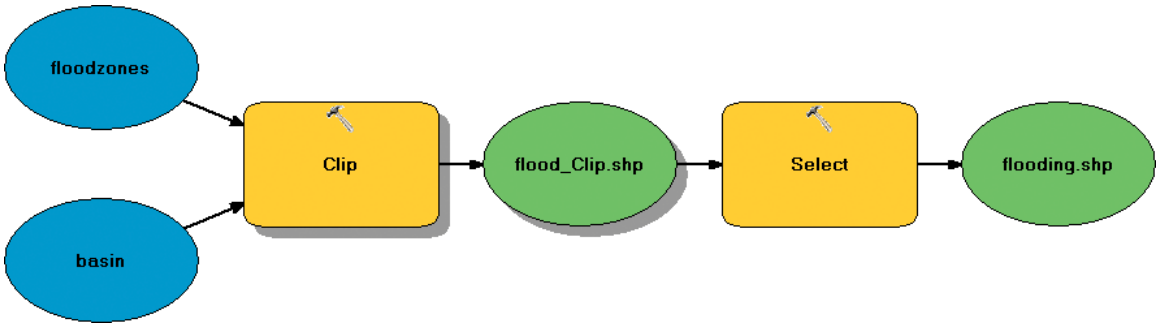


14 Click OK to close the Query Builder dialog box. The Select tool dialog box should now look like the example in the figure. ➔

15 Click OK to close the Select tool dialog box.



16 On the ModelBuilder toolbar, first click the Auto Layout button and then the Full Extent button. Your model should look like the example in the figure and is now ready to run.



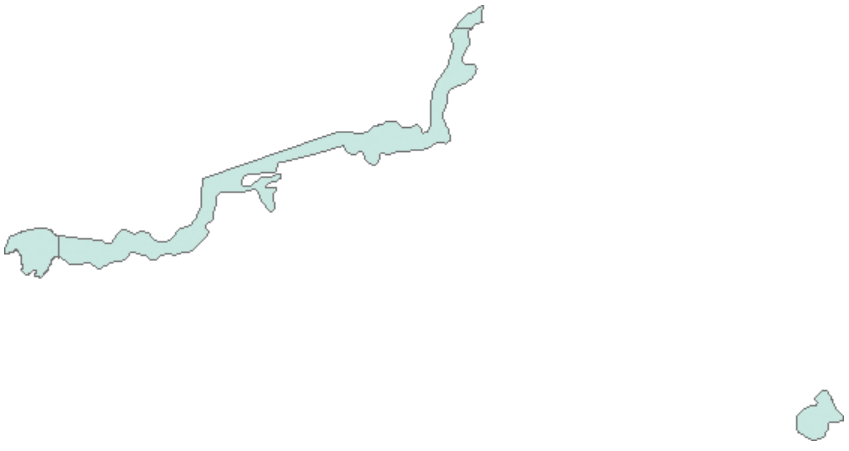
17 Right-click the flood_Clip.shp element and turn off Add To Display.

18 Right-click the flooding.shp element and turn on Add To Display.

19 On the ModelBuilder toolbar, click the Run button to run the model. Because the Clip tool was run previously, only the Select tool needs to be run for the model run to be complete.

20 When the model run is complete, close the model progress dialog box.

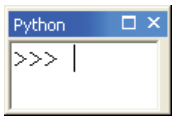
21 Save and close the model. The final result of the model is now added to the data frame.



Use scripting for geoprocessing

Scripting represents another way to carry out geoprocessing operations in ArcGIS. A basic Python script is similar to a model, except that it uses code instead of the visual programming language of ModelBuilder. Python is the preferred scripting language for working with ArcGIS, and Python code can be run directly in the Python window.

1 On the ArcMap Standard toolbar, click the Python window button .
This opens the Python window. The prompt (>>>) indicates that the Python window is ready to accept code.



To be able to run geoprocessing tools from Python, you first need to import the ArcPy site package, which you'll do next. Importing the ArcPy site package makes all the tools in the geoprocessing framework in ArcGIS available for Python scripting.

2 Following the prompt, type the following:

```
>>> import arcpy
```

Within the Python window, ArcPy is automatically referenced, so the `import arcpy` statement is in fact not necessary to use the geoprocessing tools from within that window. However, code in the Python window can be converted to a script file (.py) and stand-alone scripts do need the `import arcpy` statement.

Note: *Do not type the greater-than (>>>) symbols. They are shown here to indicate that Python code should be typed following the prompt. When a Python script is being written (which is discussed later in this section), the prompt is no longer used. In many programming environments, the prompt is referred to as a "command prompt," so you may see either term used in the documentation.*

Notice that the Python window provides prompts to assist in writing proper syntax. For example, when you start typing the letter *i*, a list is provided of the code elements that start with this letter. You can select the option you want by using the arrow keys to point to it, and then press the TAB key.

```
>>> i
```

>>> TIP

Instead of using the prompts, you can also just keep typing. Even if you don't use the prompts, they can be very useful as reminders of the proper syntax.

3 After your first line of code (`import arcpy`), press ENTER. Pressing ENTER brings up a new prompt at the next line. Remember that Python is an interpreted language, which means that in the Python window, a single line of code is run as soon as you press ENTER.

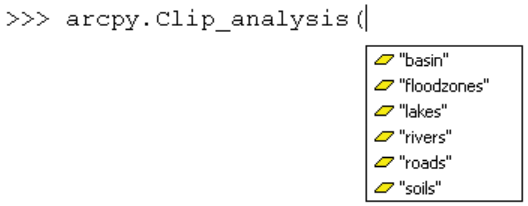
Now you are ready to run a geoprocessing tool.

4 On the next line of code, enter the following, but do not press ENTER yet:

```
arcpy.Clip_analysis
```

This code calls the Clip tool. Python is case sensitive (for the most part), so be sure to type "Clip," not "clip." Calling the Clip tool is equivalent to opening the tool dialog box. The next step is to specify the tool's parameters, as if you were filling out the tool dialog box. As you start typing, the prompts will be helpful to ensure that you use the proper syntax.

When you type an opening left paren [(] after the Clip tool, a drop-down list appears, containing all the layers from the ArcMap table of contents.

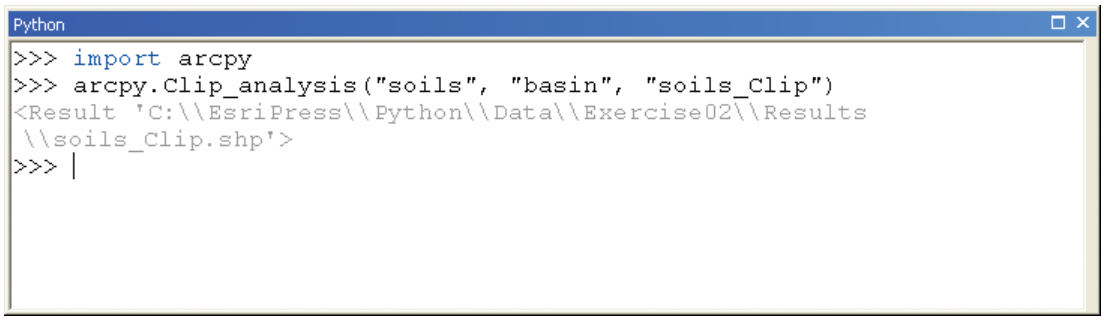


5 Complete the following line of code:

```
arcpy.Clip_analysis("soils", "basin", "soils_Clip")
```

The required tool parameters are listed inside the parens. The optional XY Tolerance is not included, which means that, just like a tool dialog box, the default value will be used.

6 **Press ENTER to run the line of code.** Similar to when a tool is run from ArcToolbox, when background processing is enabled, a progress bar appears on the ArcMap status bar, to show that the tool is running. Once the tool execution is complete, a small pop-up notification appears in the notification area, at the far right of the taskbar. The output feature class is added to the data frame, and the result is printed in the Python window.

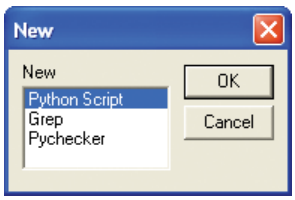


The use of Python code in the Python window is covered in more detail in chapter 3. For now, it is important to remember that you can run geoprocessing tools directly from the Python window. Lines of code are run immediately, and the Python window is highly integrated with the ArcGIS interface.

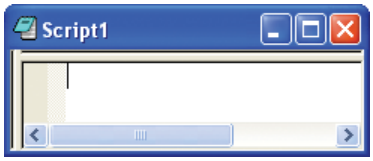
In addition to working with Python code in the Python window, you can write and run code in a Python editor. You will use the PythonWin editor in the next set of steps to create a simple script.

7 **On the taskbar, click the Start button, and then, on the Start menu, click Programs > Python 2.7 > PythonWin.** This brings up the PythonWin application.

8 **On the PythonWin menu bar, click File > New. On the New dialog box, click Python Script and click OK.**



This brings up a new script window.



9 On the menu bar, click File > Save As and save the script as my_clip.py **to the Results folder for exercise 2 (C:\EsriPress\Python\Data\Exercise02\Results\my_clip.py). Python script files are simply text files that have the .py extension.** There is no prompt (> > >) in the script window. Python code in a script is not run until the script is run. So you can enter multiple lines of code before you run the script.

10 Enter the following code in the my_clip script window:

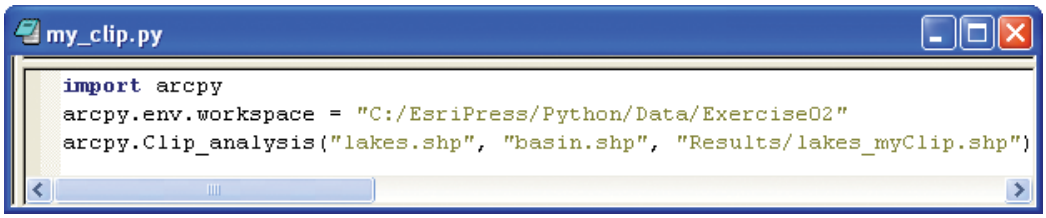
```
import arcpy
arcpy.env.workspace = "C:/EsriPress/Python/Data/Exercise02"
arcpy.Clip_analysis("lakes.shp","basin.shp", "results/lakes_myClip.➔
➔ shp")
```

Note: *This book uses an arrow symbol ➔ to indicate long lines of code that appear all on one line in Python.*

The line of code that starts with `arcpy.env.workspace` is equivalent to setting the workspace on the Environment Settings dialog box. This syntax is covered in more detail in the following chapters.

Note: *The workspace needs to be set in the Python script, even though the environment settings have been set in the geoprocessing framework in ArcMap (that is, in ArcToolbox). A stand-alone Python script does not inherit the environment settings of ArcGIS for Desktop applications.*

Your script window should now look like the example in the figure.



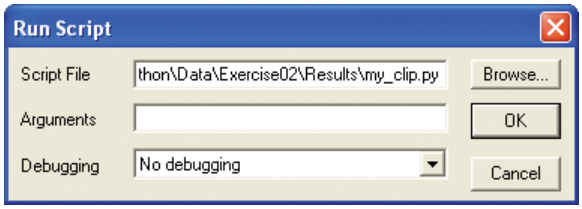
11 On the PythonWin menu bar, click File > Save to save the script.

>>> TIP

After subsequent edits, click the Run button to save the script automatically.

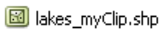
12 On the toolbar, click the Run button.

13 On the Run Script dialog box, leave the default settings (No debugging). Click OK. ➔



The script now runs. Upon execution of the script, it does not initially appear as if much has happened. Because you are running a stand-alone script, the output is not automatically added to a data frame in ArcMap. In fact, ArcMap doesn't need to be open for a script to run.

14 In ArcMap, open the Catalog window, navigate to the Results folder for exercise 2, and confirm that the lakes_myClip shapefile was created.



The Python script accomplishes the same task as the Python code in the Python window: in both places, the Clip tool runs and creates a new dataset—in this case, a shapefile. However, there are a few differences:

- The Python window inherits the environments of the geoprocessing framework in ArcMap, but in the stand-alone Python script, the environments need to be set.
- The Python script can run without having any ArcGIS for Desktop applications open, whereas the Python window is an integral part of ArcGIS for Desktop applications.
- Code in the Python window is run line by line, whereas the stand-alone Python script is run in its entirety.

These differences between running Python scripts and running code in the Python window are revisited in later chapters.

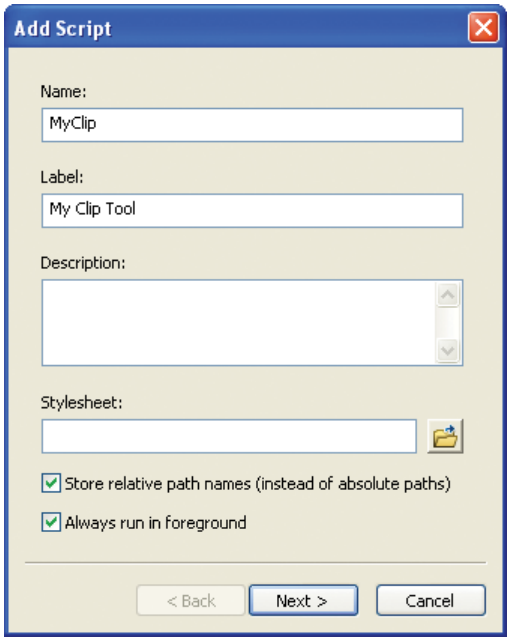
Use scripts as tools

The Python window provides a flexible environment for testing snippets of Python code, but more complex code is typically saved to a script. Python scripts can be integrated into the ArcGIS environment by adding them as tools.

1 In ArcMap, in the Catalog window, right-click the Exercise 2 Tools toolbox in the Exercise02 folder and click Add > Script.

2 On the Add Script dialog box, type MyClip **for Name and** My Clip Tool **for Label.**

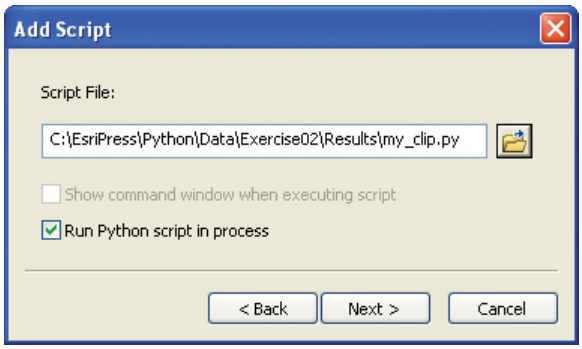
3 Select the "Store relative path names" and "Always run in foreground" check boxes.



4 Click Next.

On the next dialog box, you can select the script file that will be attached to the tool.

5 Click the Browse button and navigate to the Results folder for exercise 2. Select the my_clip.py script file.



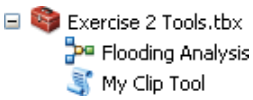
6 Click Next.

The next dialog box allows you to specify tool parameters to be displayed in the tool dialog box. You can skip this for now because the simple script you are using contains hard-coded parameters, with the

values already in place, instead of user-specified inputs. Creating script parameters for use on a tool dialog box is covered in chapter 13.

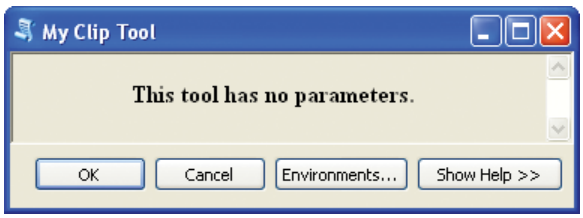
7 Leave the list of parameters blank and click Finish.

This adds a script tool to the toolbox. You can now run the script as a tool.



8 To test the script, in the Catalog window, navigate to the Results folder for exercise 2 and delete the lakes_myClip shapefile.

9 Double-click the My Clip Tool. The tool has no parameters because none were created when the script tool was set up. However, the script will run fine with the hard-coded parameters.



10 Click OK to run the tool. When the tool execution is complete, close the My Clip Tool progress dialog box.

11 In the Catalog window, right-click the Results folder for exercise 2 and click Refresh.

12 Confirm that the lakes_myClip shapefile was created.

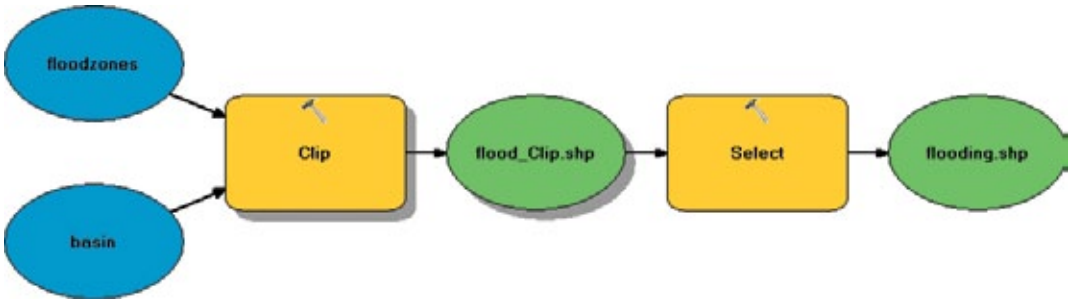
Note: *Because the script now runs as a tool, it inherits the environment settings and geoprocessing options of the current ArcMap document. For example, if you were to run the script tool again without first deleting the lakes_myClip shapefile, the tool would run fine and overwrite the existing dataset. This is because the "Overwrite the outputs of geoprocessing operations" check box was selected under Geoprocessing Options.*

As you have just seen, creating a script tool is fairly easy. However, there is much more to creating robust script tools, including setting tool parameters to obtain user input, validating input parameters, and error handling, to name a few. These topics are covered in chapter 13, after you have had more exposure to Python syntax and writing Python scripts.

Convert a model to a script

Another way to learn about Python scripting is to export a model to a script. This allows you to see what a logical sequence of geoprocessing operations looks like in Python.

For starters, you can revisit the Flooding Analysis model created earlier, as shown in the example in the figure.



You will now convert this model to a script.

1 **In the Catalog window, right-click the Flooding Analysis model in the Exercise 2 Tools toolbox in the Exercise02 folder and click Edit.**

2 **On the ModelBuilder menu bar, click Model > Export > To Python Script.**

3 **On the Save As dialog box, save the script file as** flooding.py **to the Results folder for exercise 2.**

4 **Close the model.**

5 **Return to the PythonWin application. On the Standard toolbar, click the Open button .**

6 **On the Open dialog box, browse to the Results folder for exercise 2, click the flooding.py script, and click Open.**

7 Review the contents of the script.

flooding.py

```
# -*- coding: utf-8 -*-
# ---------------------------------------------------------------------------
# flooding.py
# Created on: 2012-03-13 22:56:15.00000
#   (generated by ArcGIS/ModelBuilder)
# Description:
# ---------------------------------------------------------------------------

# Import arcpy module
import arcpy

# Local variables:
basin = "basin"
floodzones = "floodzones"
flood_Clip_shp = "C:\\EsriPress\\Python\\Data\\Exercise02\\Results\\flood_Clip.shp"
flooding_shp = "C:\\EsriPress\\Python\\Data\\Exercise02\\Results\\flooding.shp"

# Process: Clip
arcpy.Clip_analysis(floodzones, basin, flood_Clip_shp, "")

# Process: Select
arcpy.Select_analysis(flood_Clip_shp, flooding_shp, "\"SFHA\" = 'IN'")
```

Don't worry for now about being able to understand everything in the script. However, you should be able to recognize each of the model elements, including the data variables floodzones and basin, as well as the Clip and Select tools.

8 Close PythonWin.

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

In this exercise, you have learned how to run geoprocessing tools and control how they are run by using tool parameters and environment settings. You have created a model using the ModelBuilder interface. You have also run Python code in the Python window as well as running stand-alone Python scripts and script tools from within an ArcGIS for Desktop application.

Challenge exercise

Challenge 1

Create a new model called **Soil Analysis** that accomplishes the following:

1. Clips the soils layer using the basin layer

2. From the clipped version of the soils layer, selects the features that are "Not prime farmland" (field FARMLNDCL)

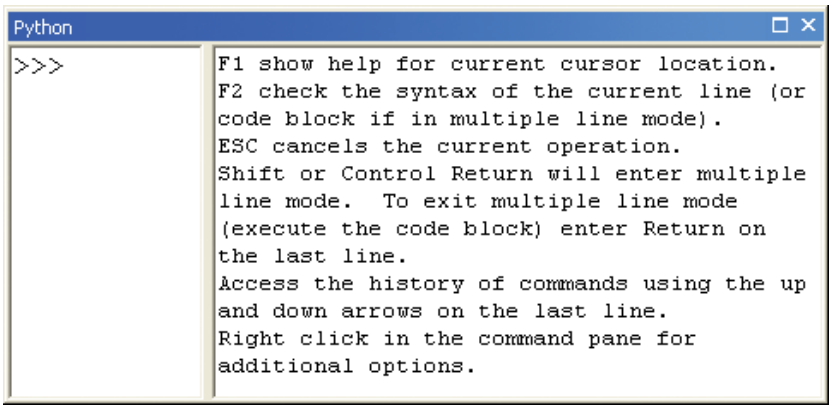
Convert the model to a script called **soil.py**.

Exercise 3
Using the Python window

Open the Python window

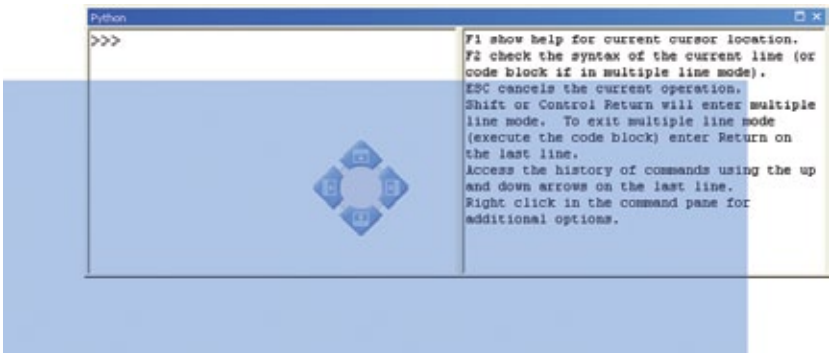
The Python window is an interactive Python interpreter and allows you to run Python code directly from within an ArcGIS for Desktop application.

1 **Start ArcMap. On the Standard toolbar, click the Python button to open the Python window.** ➔

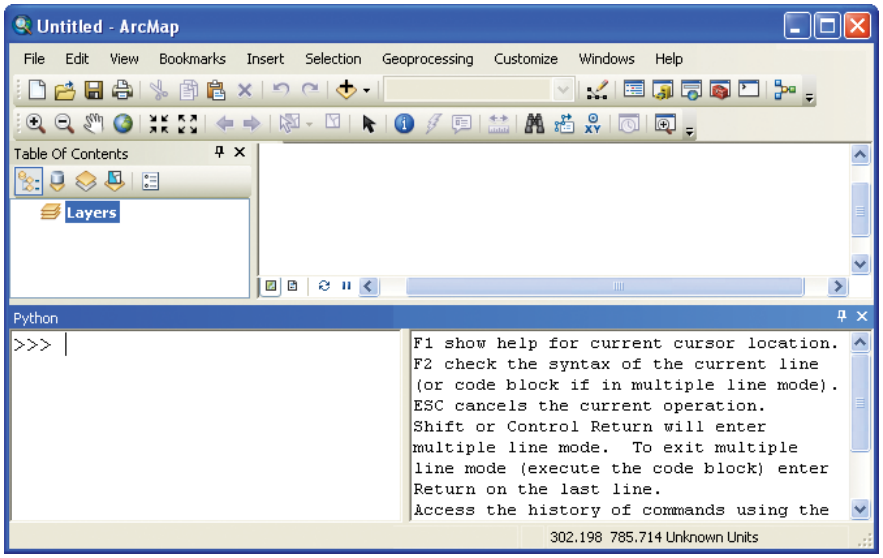


As with other windows in ArcMap, you can leave the Python window floating or dock it on any side of the ArcMap interface. Because you will be typing horizontal lines of code, it makes sense to dock the window at either the top or the bottom of the interface.

2 **To dock the Python window, drag the top bar of the Python window. This brings up eight arrows (four of which are visible here), indicating the various locations where you can dock the Python window.** ➔



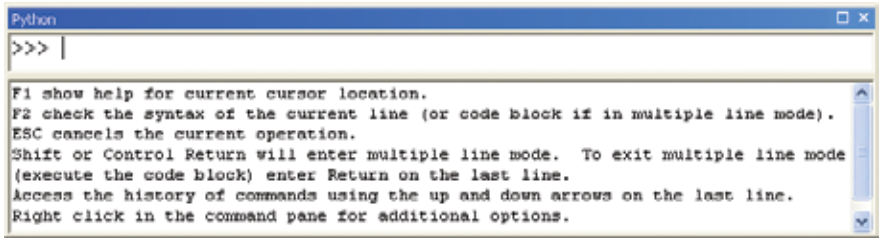
3 **Drop the Python window on the bottom arrow, so that the window is at the bottom of the interface.**



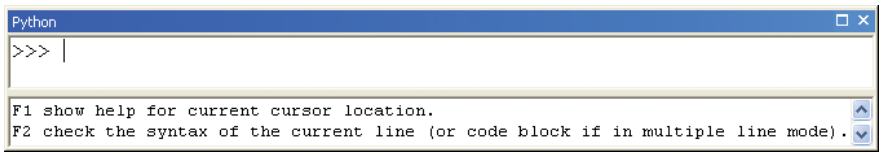
4 **To undock the Python window, drag the top bar again.**

The Python window itself can be resized, and the divider between the code section and the Help and syntax panel can be moved. Placement of the Help and syntax panel can also be controlled.

5 **Right-click in the Python window and click Help Placement > Bottom.**



6 **Drag the divider between the code section and the Help and syntax panel to adjust their sizing the way you like it.**



>>> TIP

Docking the window makes it a bit easier to keep working with your code and manage your layers in the ArcMap table of contents at the same time.

Write and run code

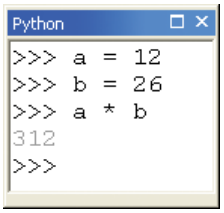
As in any interactive interpreter, Python code is run one line at a time, and the results are printed immediately.

1 Type the following code and press ENTER at the end of each line:

```
>>> a = 12
>>> b = 26
>>> a * b
```

Note: *Whether you use spaces here or not does not influence the execution of the code—that is,* `a = 12` *is the same as* `a=12`*. Spaces are commonly used in Python to make code easier to read but are often not required.*

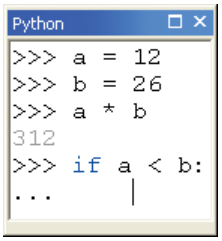
After you press ENTER, the line of code is executed, and the next line automatically starts with a new command prompt. The result of the preceding code is shown in the figure. ➔



Now you can try something different.

2 At the command prompt, type the following code:

```
>>> if a < b:
```



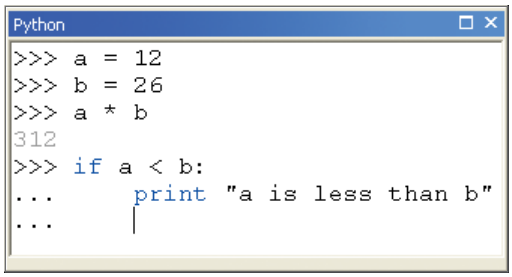
3 Press ENTER to run the line of code. The result is a secondary prompt at the next line, consisting of three dots (...). ➔

It means that the interactive Python interpreter recognized the start of the multiline construct. The `if` statement is the first line of a block of code, and at least one more line of code is needed for the code to run successfully. The Python window automatically indents the next line of code.

4 At the secondary prompt (...), enter the following code. Because this is the block of code following the `if` statement, the code has been indented. (This indentation was not automatic in ArcGIS 10.0 and had to be added manually by typing four spaces.)

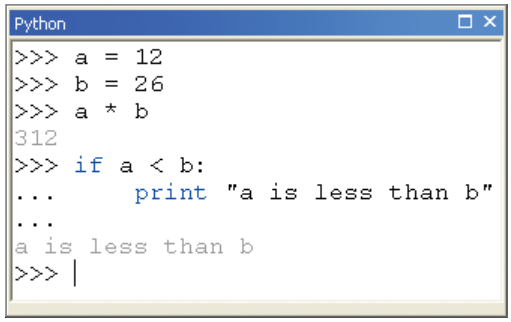
```
... print "a is less than b"
```

5 **At the end of the line of code, press ENTER.** Notice the result. The line of code is not executed, and the next line starts with another secondary prompt. ➔



6 **With the pointer at the start of the next line of code, and without entering any code, press ENTER.** ➔

When you are working with the secondary prompt, code is executed only when you press ENTER twice. This runs all lines of code following the last primary prompt.

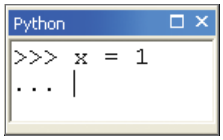


The Python window provides the secondary prompt automatically when it recognizes a multiline construct. However, you can also force the secondary prompt by pressing CTRL + ENTER.

7 **Right-click in the Python window and click Clear All.** This removes all lines of code from the Python window.

8 **At the primary prompt, enter the following code:**

```
>>> x = 1
```



9 **At the end of the line of code, press CTRL+ENTER.** ➔

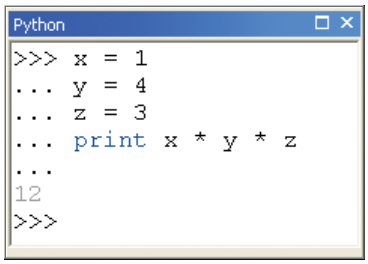
This brings up the secondary prompt. You can keep entering lines of code and pressing ENTER at the end of each line—the secondary prompt continues to appear, and the code is not executed until you press ENTER twice.

10 **At the secondary prompt, enter the following code:**

```
... y = 4
... z = 3
... print x * y * z
```

Note: *There is no need to enter spaces here following the secondary prompt. In an earlier example using the secondary prompt, spaces were needed to create a block of indented code, but this is not the case here.*

11 At the end of the last line of code, press ENTER twice. ➔



The use of CTRL+ENTER makes it possible to complete several lines of code before running it.

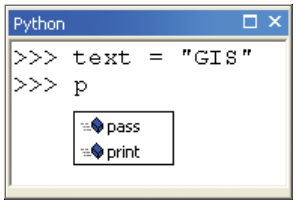
As you were typing the `print` and `if` statements, you probably already noticed the code autocompletion prompts. Next, you will take a closer look at how these prompts work.

12 Right-click in the Python window and click Clear All.

13 At the primary prompt, enter the following code and press ENTER:

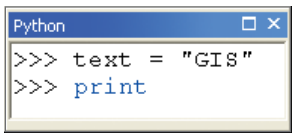
```
>>> text = "GIS"
```

14 At the next line, start typing the `print` statement. ➔



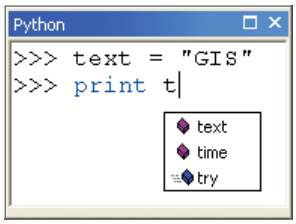
As soon as you start typing the letter *p*, you are prompted by a list of suggestions. These code autocompletion prompts include any text that would be logical based on Python syntax. In this case, there are two Python statements that start with the letter *p*. You can select the option you want using your pointer or by using the UP ARROW and DOWN ARROW keys.

15 Select the `print` statement and press the TAB key. ➔



For a statement as short as `print`, using the prompts does not save much typing, but autocompletion prompts can be very helpful as reminders of the proper syntax, and they can help you avoid making typos.

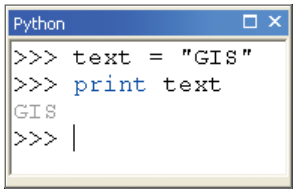
16 After the `print` statement, type a space, followed by the letter *t*. ➔



Notice that the list of suggestions is not limited to built-in Python terms but also includes `text` since this was used earlier in the code.

Note: *The term* `text` *is a variable here, a term that is covered in chapter 4.*

17 Select the `text` variable and press the TAB key. Then press ENTER to run the code. ➔



It is worthwhile to note that you do not have to use the code autocompletion prompts. Instead of selecting one of the suggested terms, you can simply continue typing. You can also turn off the prompts by right-clicking in the Python window and selecting or clearing the Show Default Choices option. However, you can benefit from code

autocompletion prompts, because using them reduces typos, makes writing code faster, and shows all your options in the drop-down list.

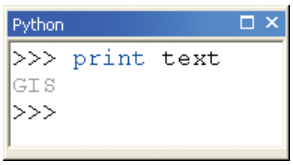
You have already seen how to clear code: right-click in the Python window and click Clear All. However, you can also continue to run lines of code, and the window will start to scroll downward if the lines of code do not fit in the window. Also, clearing the lines of code does not refresh the interactive Python interpreter—rather, the code executed earlier is still in memory.

18 Right-click in the Python window and click Clear All.

19 At the command prompt, type the following code and press ENTER:

```
>>> print text
```

Notice that the code printed the correct text, as shown in the figure, even though the lines of code are no longer visible. ➔



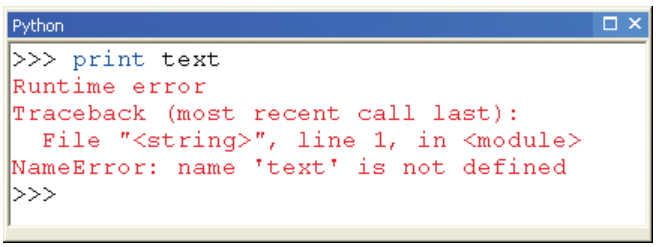
Closing and opening the Python window does not remove the code, nor does it remove the code from the interactive Python interpreter. However, closing ArcMap removes the code from memory, and you can never use it again.

20 Close ArcMap and click No if asked to save any changes to the map document. Start ArcMap again. Open the Python window if necessary.

21 At the command prompt, type the following code and press ENTER:

```
>>> print text
```

Notice the result, as shown in the figure. ➔



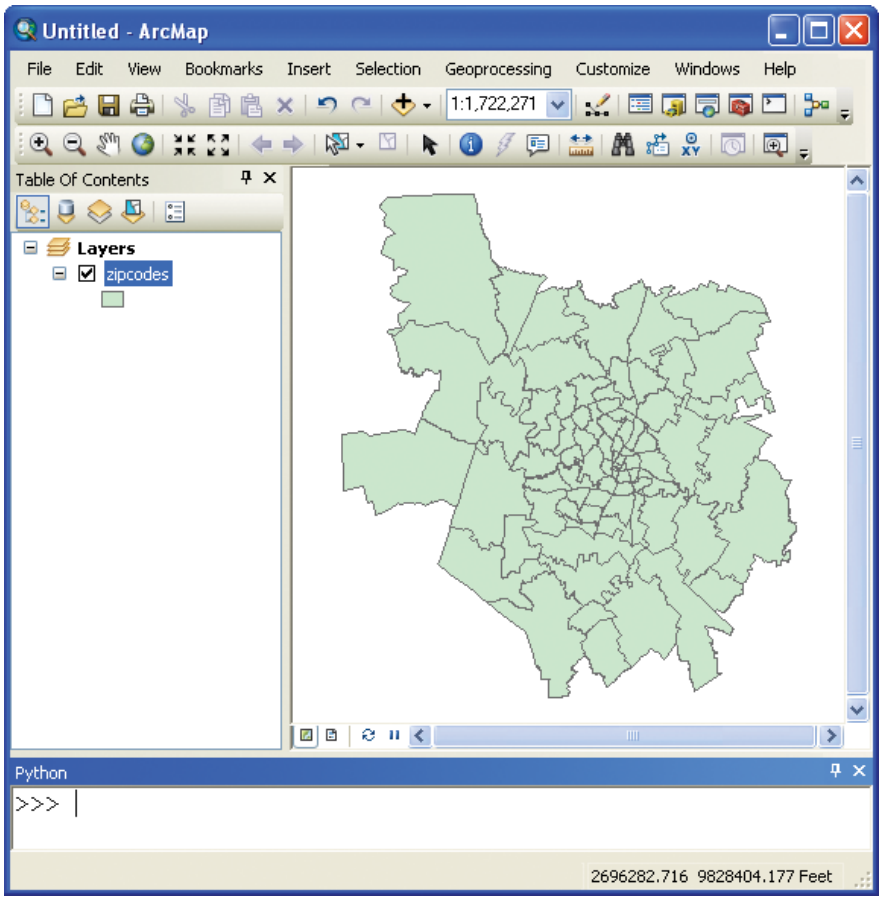
The variable text is not defined and therefore the Python window produces an error message. Code in the Python window is removed from memory when the ArcGIS for Desktop application is closed. Later in this exercise, you will see that there is an option to save the Python code.

Run a geoprocessing tool

Next, you will run a geoprocessing tool from the Python window that works with a layer in ArcMap. Don't worry too much about the syntax of the code for now.

1 **On the ArcMap Standard toolbar, click Add Data and browse to the C:\EsriPress\Python\Data\Exercise03 folder.**

2 **Select the zipcodes.shp file and click Add.**



3 **At the prompt, start typing the following:**

```
>>> count = arcpy.G
```

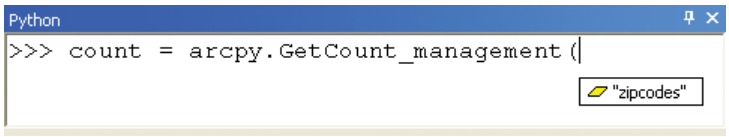
The code prompts you by providing a list of options.

4 **Select the `GetCount_management` option from the list and press the TAB key.**

5 Type an opening, left paren [(]:

```
>>> count = arcpy.GetCount_management(
```

The code prompts you by providing the name of the only layer in ArcMap.



6 Select this layer, press the TAB key, and type a closing, right paren [)]:

```
>>> count = arcpy.GetCount_management("zipcodes")
```

7 Press ENTER to run the line of code. The code runs the Get Count tool. Once it is finished running, a pop-up notification appears in the notification area, at the far right of the taskbar. Geoprocessing messages also appear in the Help and syntax panel of the Python window.

Note: If you have disabled background processing under Geoprocessing Options, no pop-up notification will appear and messages appear only in the Help and syntax panel of the Python window.

8 At the prompt, enter the following code and press ENTER:

```
>>> print count
```

The code prints the result of the Get Count tool: 80.

The specific syntax used in this example is covered in detail in chapter 5. For now, the important thing to remember is that you can run code in the Python window that interacts with spatial data in the map document as well as on disk.

Get help in the Python window

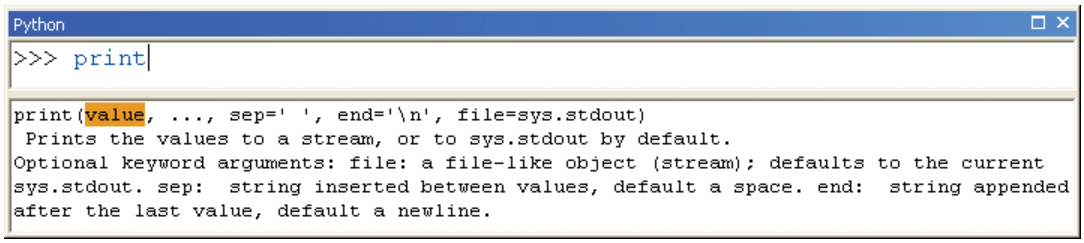
You have already seen several examples of code autocompletion prompts, which can be of great help. These prompts are context sensitive, meaning that the suggestions include only terms that are logical based on Python syntax.

There are several other ways to get assistance.

1 Right-click in the Python window and click Clear All.

2 Make sure the Help and syntax panel is visible. →

3 At the command prompt, type the `print` statement.

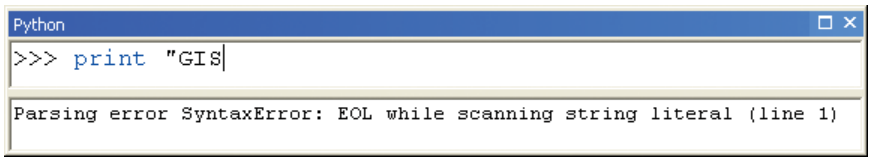


What is shown in the Help and syntax panel is the syntax for the `print` statement. When you are just getting started in Python, the wording may appear a little cryptic.

4 Continue the line of code with the following:

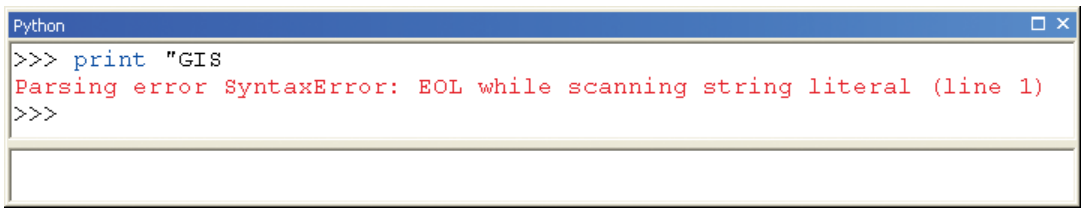
```
>>> print "GIS
```

5 With the pointer at the end of the line of code, press F2.



This brings up syntax checking for the current line of code. In this case, an end-of-line (EOL) error is detected. Pressing F2 effectively prints any syntax errors that will occur when the line of code is executed.

6 Without fixing the syntax error, press ENTER to run the line of code.

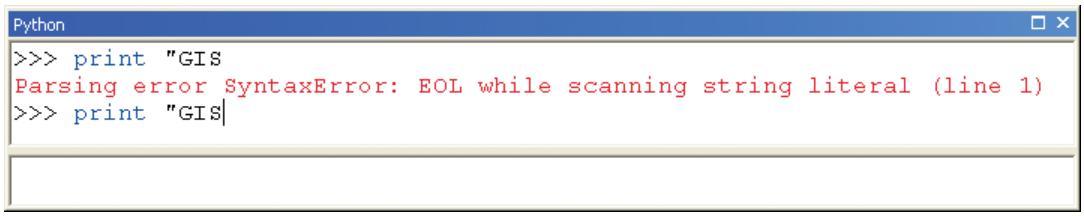


Notice that this is the same error as reported using syntax checking. The F2 key does only syntax checking—other types of errors are discovered only when a tool is actually run.

If you make an error, you may be tempted to correct prior lines of code that have already been run. However, you can only run code at the current command prompt, so fixing prior lines of code is not an option. To save on typing, you can copy the line of code, paste it after the current

command prompt, and fix the error. Since this is such a common task, there are built-in shortcuts to make it easier.

7 At the command prompt, press the UP ARROW key to bring up the previous line of code.



Now you can copy and paste the line of code, fix it, and then run it. The UP ARROW and DOWN ARROW keys can be used to scroll up or down to any previous line of code in the current session.

Save your work

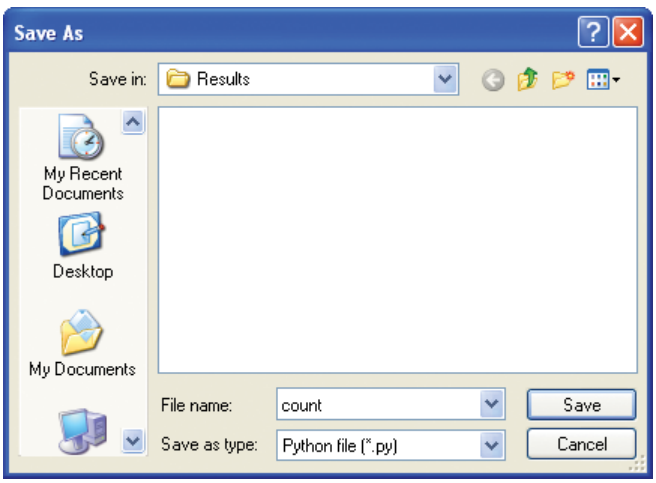
Code in the Python window is not saved with a map document. If you want to reuse your code later, you can save the code to a script.

1 Right-click in the Python window and click Clear All.

2 At the command prompt, enter and run the following lines of code.

```
>>> count = arcpy.GetCount_management("zipcodes")
>>> print count
80
```

You can copy and paste lines of code from the Python window to a text editor or to a script window of a Python editor to save and reuse. However, this also copies text that is not code, such as prompts, results, and messages.



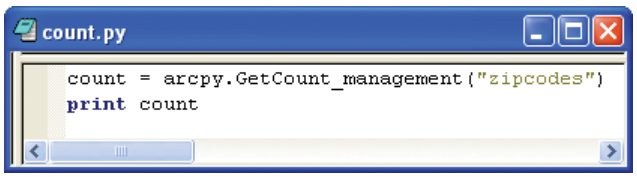
3 Right-click in the Python window and click Save As.

4 On the Save As dialog box, browse to the C:\EsriPress\Python\Data\Exercise03\Results folder and save your file as count.py. ➔

Next, you can open the script file in a Python editor.

5 **Start PythonWin.**

6 **On the Standard toolbar, click the Open button, browse to the Results folder for exercise 3, select the count.py script, and click Open.**



Notice that the resulting script file contains code only.

7 **Close PythonWin.**

Load existing code

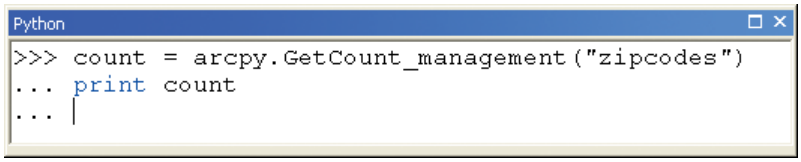
You can also load existing code into the Python window.

1 **Return to ArcMap.**

2 **Right-click in the Python window and click Clear All.**

3 **Right-click in the Python window and click Load.**

4 **On the Open dialog box, browse to the Results folder for exercise 3, select the count.py script, and click Open.**



The Python script loads into the Python window. Notice that the lines of code are preceded by the secondary prompt, meaning that the code is not run line by line. You can now make changes to the code prior to running it.

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

Exercise 4

Learning Python language fundamentals

Work with numbers

Python can be used as a powerful calculator. Practicing math calculations in Python will help you not only perform these tasks, but also show you how Python works with different types of numbers.

1 **Start ArcMap. On the Standard toolbar, click the Python button.**

2 **In the Python window, type the following code and press ENTER:**

```
>>> 12 + 17
```

Note: Do not type the prompt (>>>), since it is already provided by the Interactive Window. Whenever sample code in this exercise is preceded by the prompt, it means the code should be entered in the Interactive Window.

This code should give you the result 29. All basic calculator functions work just as you would expect, with one exception, which follows.

3 **Run the following code:**

```
>>> 10 / 3
```

And the result is 3? What went wrong? The inputs to the calculation (10 and 3) are both integers, and therefore the result is, by default, also an integer. This results in rounding. If you want ordinary division, the solution is to use real numbers or floats—that is, numbers with decimals. If either one of the inputs in a division is a float, the result will also be a float.

4 Run the following code:

```
>>> 10.0 / 3.0
```

The result is 3.3333333333333335—notice that the very last number does not make sense because it has reached the limit of the number of decimal places Python uses.

Note: The exact number of decimal places depends on the version of Python you are using.

Is there a similar upper limit to the size of integers? Yes, ordinary integers cannot be larger than 2147483647 or smaller than -2147483647. However, if you want larger values, you can use a long integer, commonly referred to as "long."

5 Run the following code:

```
>>> 12345678901
```

The result is 12345678901L, with the letter L indicating that Python converted the input value to a long integer.

Basic arithmetic operations such as addition, subtraction, multiplication, and division are relatively straightforward. Many more operations are possible, but take a look at just one more for now: the exponentiation, or power, operator (**).

6 Run the following code:

```
>>> 2 ** 5
```

And the result is 32.

Although you are not very likely to use Python directly as a calculator, the examples here show you how Python handles numbers, which will be useful as you start writing scripts.

Work with strings

Now you can take a look at strings. You have already seen a very simple example, which follows.

1 Run the following code:

```
>>> print "Hello World"
```

The code prints `Hello World` to the next line. This is called a *string*, as in a string of characters. Strings are values, just as numbers are.

Python considers single and double quotation marks the same, making it possible to use quotation marks within a string.

2 Run the following code:

```
>>> print 'Let's go!'
```

The code results in a syntax error because Python does not know how to distinguish the quotation marks that mark the beginning and end of the string from the quotation marks that are part of the string—in this case, in the word "Let's." The solution is to mix the type of quotation marks used, with single and double quotation marks.

3 Run the following code:

```
>>> print "Let's go!"
```

The code prints `Let's go!` to the next line.

Strings are often used in geoprocessing scripts to indicate path and file names, so you will see more examples of working with strings throughout the exercise.

Strings can be manipulated in a number of ways, as you will see next.

4 Run the following code:

```
>>> z = "Alphabet Soup"
>>> print z[7]
```

The code returns the letter *t*, the seventh letter in the string where the letter A is located at index number 0. This system of numbering a sequence of characters in a string is called *indexing* and can be used to fetch any element within the string. The index number of the first element is 0.

5 Run the following code:

```
>>> print z[0]
```

The code returns the letter A. The index number of the last element depends on the length of the string itself. Instead of determining the length, negative index numbers can be used to count from the end backward.

Note: *Quotation marks in Python are "straight up," and there is no difference between opening quotation marks and closing quotation marks, as is common in word processors. When you type quotation marks directly in Python, they are automatically formatted properly, but be careful when copying and pasting from other documents. Quotation marks in Python have to look like this (' ') or this (" "), not like this (‘ ’) or this (“ ”).*

6 **Run the following code:**

```
>>> print z[-1]
```

The code returns the letter *p*.

To fetch more than one element, you can use multiple index numbers. This is known as *slicing*.

7 **Run the following code:**

```
>>> print z[0:8]
```

The reference `z[0:8]` returns the characters with index numbers from 0 up to, but not including, 8, and therefore the result is Alphabet.

As you have seen, you can use an index to fetch an element. You can also search for an element to obtain its index.

8 **Run the following code:**

```
>>> name = "Geographic Information Systems"
>>> name.find ("Info")
```

The result is 11, the index of the letter I. In this example, `find` is a method that you can use on any string. Methods are explored later in this exercise.

Work with variables

All scripting and programming languages work with variables. A variable is basically a name that represents or refers to a value. Variables store temporary information that can be manipulated and changed throughout a script. Many programming languages require that variables be declared before they can be used. Declaring means that you first create a variable and specify what type of variable it is—and only then can you actually assign a value to that variable. In Python, you immediately assign a value to a variable (without declaring it), and from this value, Python then determines the nature of the variable. This typically saves a lot of code and is one reason why Python scripts are often much shorter than code in other programming languages.

Next, you can try a simple example using a numeric value.

1 **Run the following code:**

```
>>> x = 12
>>> print x
```

The value of 12 is now printed to the next line. The code line `x = 12` is called an *assignment*. The value of 12 is assigned to the variable *x*. Another way of putting this is to say that the variable *x* is bound to the value of 12. Implicitly, this particular line of code results in variable *x* being an integer, but there is no need to explicitly state this with extra code.

Once a value is assigned to a variable, you can use the variable in expressions, which you'll do next.

2 **Run the following code:**

```
>>> x = 12
>>> y = x / 4
>>> print y
```

The result is 3.

Variables can store many different types of data, including numbers (integers, longs, and floats), strings, lists, tuples, dictionaries, files, and many more. So far, you have seen only integers. Next, you can continue with strings.

3 **Run the following code:**

```
>>> k = 'This is a string'
>>> print k
```

Note: *Variable names can consist of letters, digits, and underscores (_). However, a variable name cannot begin with a digit.*

Work with lists

Lists are a versatile Python data type used to store a sequence of values. The values themselves can be numbers or strings.

1 **Run the following code:**

```
>>> w = ["Apple", "Banana", "Cantaloupe", "Durian", "Elderberry"]
>>> print w
```

This prints the contents of the list.

Lists can be manipulated using indexing and slicing techniques, very much like strings.

2 Run the following code:

```
>>> print w[0]
```

This returns `Apple` because the index number of the first element in the list is 0. You can use negative numbers for index positions on the right side of the list.

3 Run the following code:

```
>>> print w[-1]
```

This returns `Elderberry`.

Slicing methods using two index numbers can also be applied to lists, which you'll try next.

4 Run the following code:

```
>>> print w[2:-1]
```

The reference `w[2:-1]` returns the elements from index number 2 up to, but not including, -1, and therefore the result is `['Cantaloupe', 'Durian']`.

Notice the difference here between indexing and slicing. Indexing returns the value of the element, and slicing returns a new list. This is a subtle but important difference.

Use functions

A function is like a little program you can use to perform a specific action. Although you can create your own functions, Python has functions already built in, referred to as *standard functions*.

1 Run the following code:

```
>>> d = pow (2, 3)
>>> print d
```

So instead of using the exponentiation operator (**), you can use a power function called `pow`. Using a function this way is referred to as

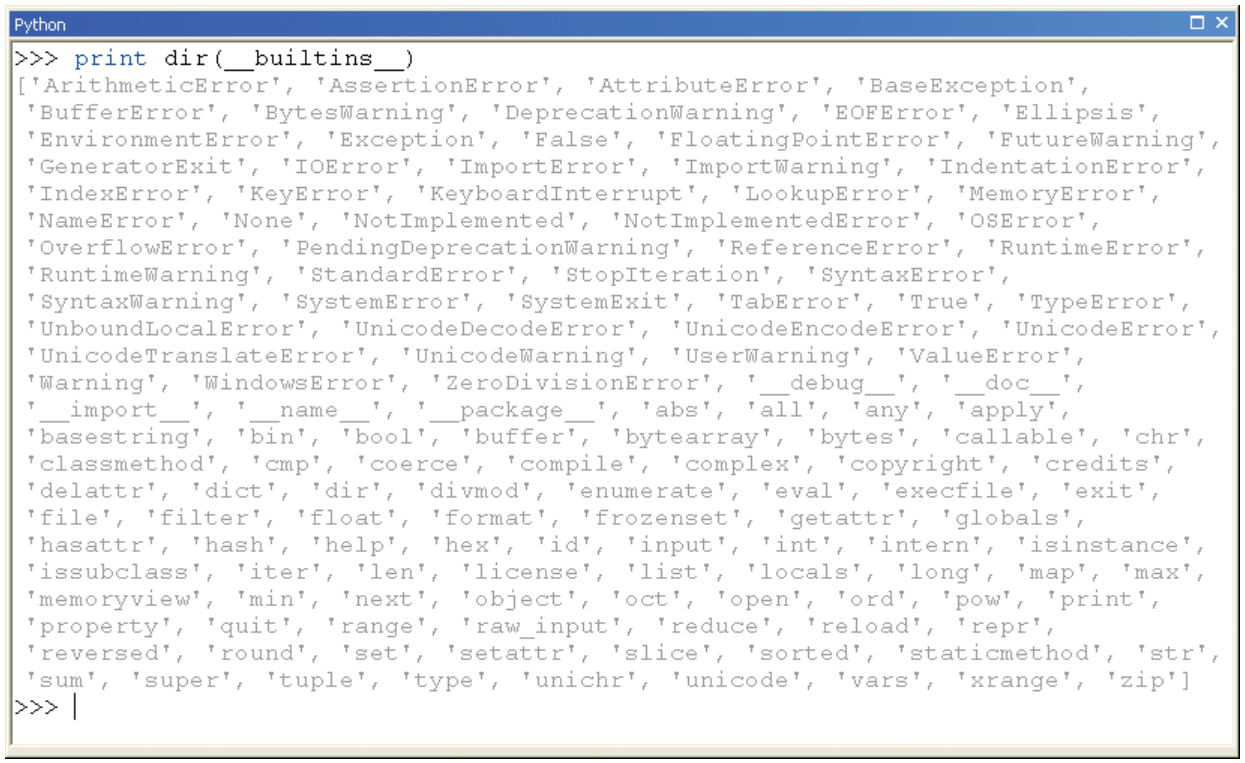
calling the function. You supply the function with *parameters*, or *arguments* (in this case, 2 and 3), and it *returns* a value.

Numerous standard functions are available in Python. You can view the complete list by using the `dir(__builtins__)` statement.

__Note:__ There are two underscores on either side of the word "builtins," not just one.

2 Run the following code:

```
>>> print dir(__builtins__)
```



It may not be immediately intuitive as to what many of these functions are used for, although some are straightforward. For example, `abs` returns the absolute value of the numeric value.

3 Run the following code:

```
>>> e = abs(-12.729)
>>> print e
```

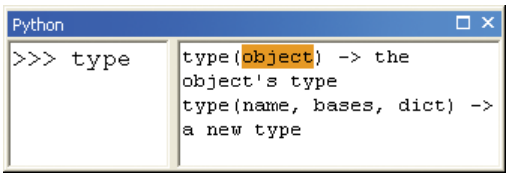
This returns the value of 12.729.

Since it may not be immediately clear how many of these functions work and what the parameters are, it should be helpful to take a look at the Help function next.

4 **First, clean up the Python window by removing all the code so far. Right-click in the Python section of the window and click Clear All.**

5 **Make sure the Help and syntax panel is visible by dragging the divider in place.**

6 **Type the function `type` and don't press ENTER yet. Notice that the syntax appears in the adjacent panel.** ➔



You can find similar descriptions in the Python manuals, but having it right where you are coding in the Python window is convenient. Notice that a section of the syntax is highlighted. It specifies the parameters of the function. When you call the function, you need to supply these parameters for the function to work, although some parameters are optional.

Next, you can try out this function.

7 **Run the following code:**

```
>>> type(123)
```

The result is `<type 'int'>`—that is, the input value is an integer.

8 **Run the following code:**

```
>>> type(1.23)
```

The result is `<type 'float'>`—that is, the input value is a float, or floating point.

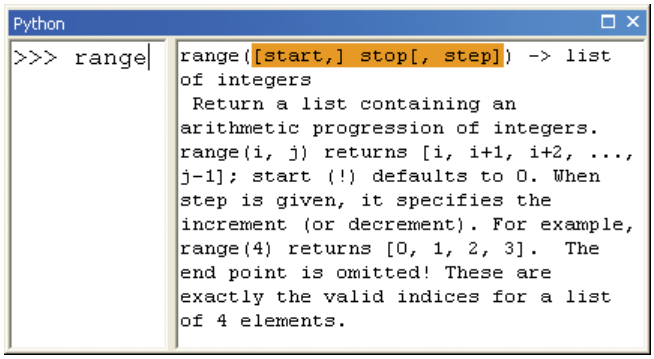
9 **Run the following code:**

```
>>> type("GIS")
```

The result is `<type 'str'>`—that is, the input value is a string.

Multiple parameters are separated by commas. Optional parameters are shown between square brackets ([ ]). For example, take a look at the function `range`.

10 In the Python window, type the function `range` and notice that the syntax Help appears in the adjacent panel.



Notice the syntax: `range([start,] stop[, step])`. The function `range` has three parameters: `start`, `stop`, and `step`.

11 Run the following code:

```
>>> range(10, 21, 2)
```

The result is `[10, 12, 14, 16, 18, 20]`.

The function returns a list of integers from 10 to 20, with an increment of 2. However, the only required parameter is the endpoint (`stop`).

12 Run the following code:

```
>>> range(5)
```

The result is `[0, 1, 2, 3, 4]`.

The function returns a list of integers using the default values of 0 for the start parameter and 1 for the increment (`step`) parameter.

Use methods

Methods are similar to functions. A method is a function that is closely coupled with an object—for example, a number, a string, or a list. In general, a method is called as follows:

```
<object>.<method>(<arguments>)
```

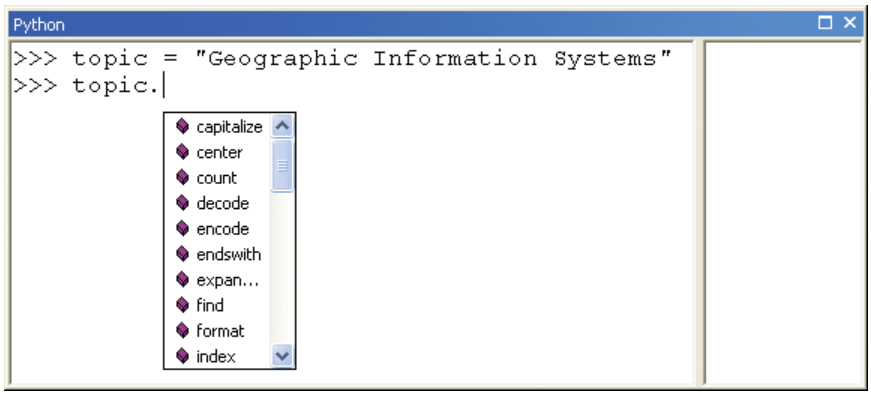
Calling a method looks just like calling a function, but now the object is placed before the method, with a dot (.) separating them. Next, take a look at a simple example.

1 **Run the following code:**

```
>>> topic = "Geographic Information Systems"
>>> topic.count("i")
```

The code returns the value of 2 because that is how often the letter *i* occurs in the input string.

A number of different methods are available for strings. Notice that when you start calling methods by typing a dot after the variable, a list of methods is provided for you to choose from. The syntax Help is also context sensitive.



Next, try this out by using the `split` method.

2 **Run the following code:**

```
>>> topic.split(" ")
```

The result is a list of the individual words in the string:

```
['Geographic', 'Information', 'Systems']
```

Next, you can see how to apply the `split` method to work with paths. Say, for example, the path to a shapefile is c:\data\part1\final. How would you obtain just the last part of the path?

3 Run the following code:

```
>>> path = "c:/data/part1/final"
>>> pathlist = path.split("/")
>>> lastpath = pathlist[-1]
>>> print lastpath
```

The result is `final`.

So what happened exactly? In the first line of code, the path is assigned as a string to the variable `path`. In the second line of code, the string is split into four strings, which are assigned to the list variable `pathlist`. And in the third line of code, the last string in the list with index -1 is assigned to the string variable `lastpath`.

Methods are also available for other objects, such as lists.

4 Run the following code:

```
>>> mylist = ["A", "B", "C"]
>>> mylist.append("D")
>>> print mylist
```

The result is `['A', 'B', 'C', 'D']`.

Very few built-in methods are available for numbers, so in general, you can use the built-in functions of Python or import the `math` module (see next section) to work with numeric variables.

Use modules

Hundreds of additional functions are stored in modules. Before you can use a function, you have to import its module using the `import` function. The functions you used in the preceding sections are part of Python's built-in functions and don't need to be imported. One of the most common modules to import is the `math` module, so you'll start with that one.

1 Run the following code:

```
>>> import math
>>> h = math.floor (7.89)
>>> print h
```

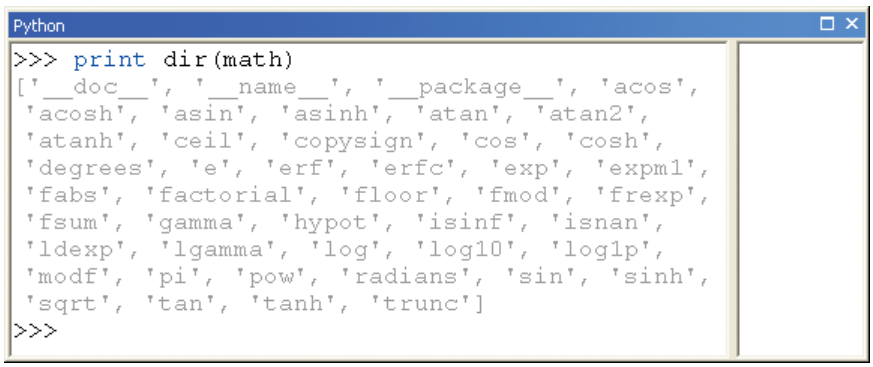
The result is 7.0.

Notice how the `math` module works: you import a module using `import`, and then use the functions from that module by writing `<module>.<function>`. Hence, you use `math.floor`. The `math.floor` function always rounds down, whereas the built-in `round` function rounds to the nearest integer.

You can obtain a list of all the functions in the `math` module using the `dir` statement.

2 Run the following code:

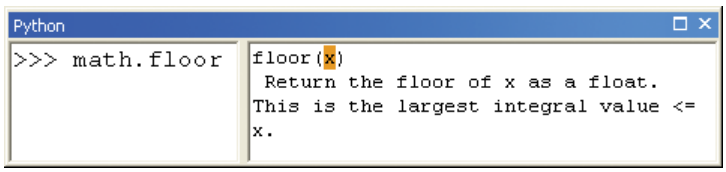
```
>>> print dir(math)
```



You can learn about each function in the Python manuals, but remember that you can also see the syntax in the Python window's Help and syntax panel.

3 Type the following code and do not press ENTER:

```
>>> math.floor
```

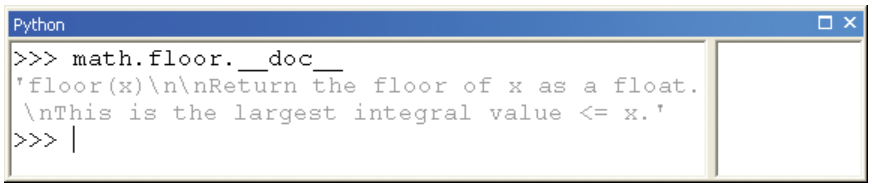


Another way to see the documentation is to use the `__doc__` statement directly in Python.

4 Run the following code:

```
>>> print math.floor.__doc__
```

The result is a printout of the same syntax Help but now directly within the interactive Python interpreter.

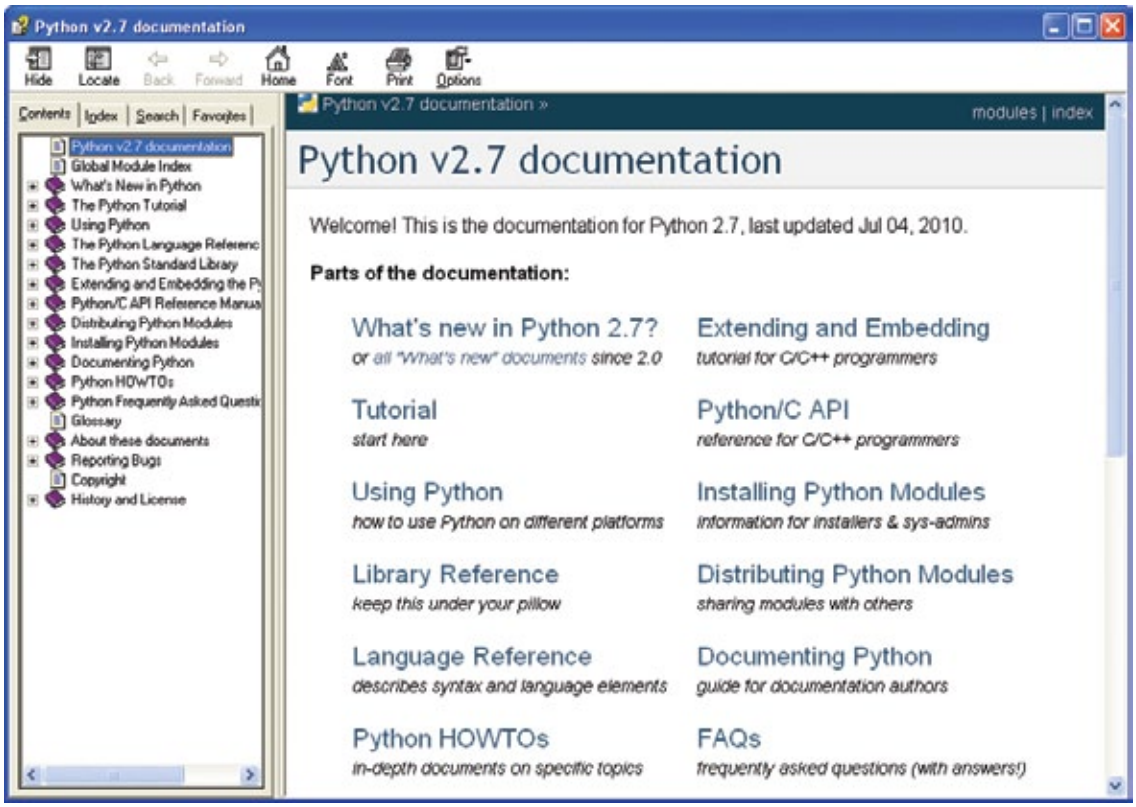


The syntax is `floor(x)`, which means the only parameter of this function is a single value. The function returns a float (such as 7.0), not an integer (such as 7). This information allows you to determine whether the function is really what you are looking for and how to use it correctly.

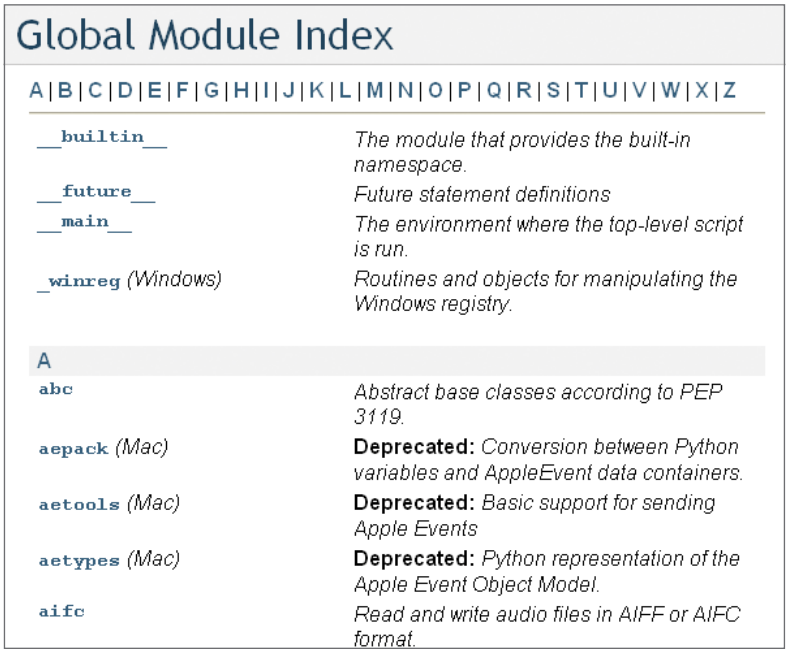
There are numerous modules available in Python. A complete list can be found in the Python manuals, which you'll look at next.

Note: *Remember that you need to add a prefix to any non-built-in functions using the name of the module, as in* `math.floor(x)`*, not just* `floor(x)`*. If you do not use a prefix, you will get an error stating that the name* `floor` *is not defined.*

5 **To access the Help documentation, on the taskbar, click the Start button, and then, on the Start menu, click All Programs > ArcGIS > Python 2.7 > Python Manuals.**



6 **In the documentation table of contents, under "Indices and tables," click Global Module Index. The index provides an alphabetical list of all the available modules.** ➔



There are many specialized modules, and in a typical Python script, you may use several. Try taking a look at just one more. For example, scroll down to the `random` module and click the link. Scroll down to the `uniform` function and read the description.

`random.` **`uniform`**(*a*, *b*)

Return a random floating point number *N* such that `a <= N <= b` for `a <= b` and `b <= N <= a` for `b < a`.

Notice that the `uniform` function has two required parameters, a and b. You will try this function next in Python.

7 Close the documentation and return to the Python window.

8 Run the following code:

```
>>> import random
>>> j = random.uniform(0, 100)
>>> print j
```

The result is a float from 0 to 100.

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

Save Python code as scripts

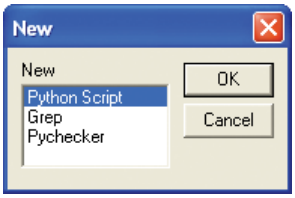
So far in this exercise, you have only worked from within the Python window, or interactive Python interpreter. This works great to practice writing Python code and to run relatively simple code. However, once code gets a bit more complex, you'll typically want to save your work to a Python script. Although you can save your code from the Python window to a script file, you will first practice creating, writing, and saving scripts using the PythonWin editor.

1 On the taskbar, click the Start button, and then, on the Start menu, click Python 2.7 > PythonWin. Notice that, by default, PythonWin opens with an Interactive Window, which works very much like the Python window in ArcMap.

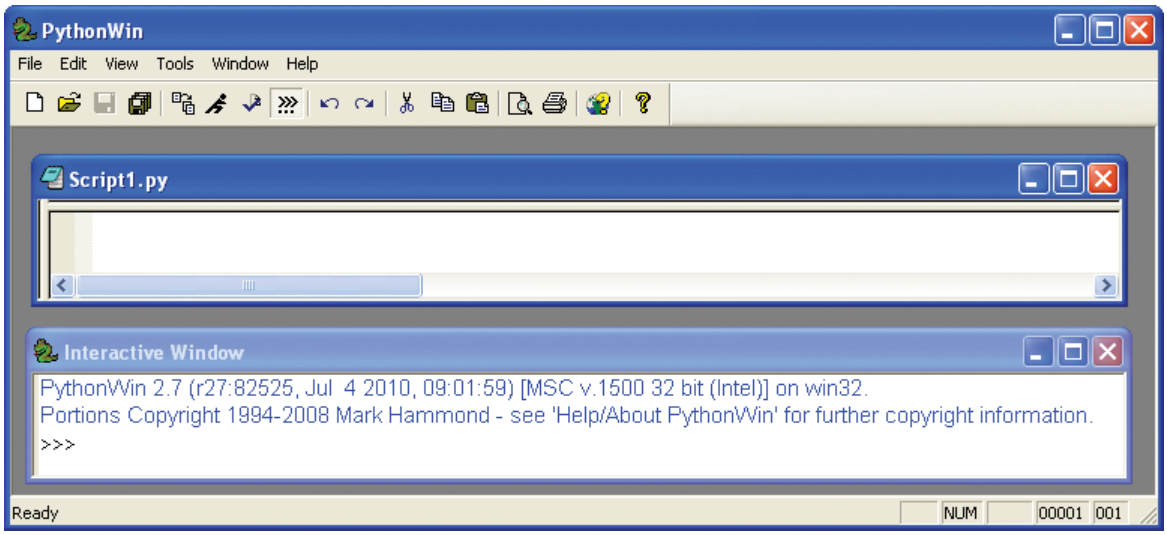
Next, you will create a new script window.

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

Note: Using the New button in PythonWin is the same as using File > New from the menu bar.



3 Rearrange the windows so that both the Interactive Window and the new script window are visible.



Next, you will enter the same code you worked with in the Python window.

4 In the script window, type the following code:

```
path = "c:/data/part1/final"
pathlist = path.split("/")
lastpath = pathlist[-1]
print lastpath
```

Notice that the lines of code in the script window are not preceded by the prompts (>>>) found in the Interactive Window. Also notice that nothing is printed when you press ENTER—the cursor simply jumps to the next line as in a text editor. What this means is that in the script window, the Python code is not actually executed until you run it.

5 First, save the script. On the PythonWin Standard toolbar, click the Save button . On the Save As dialog box, navigate to the C:\EsriPress\Python\Data\Exercise04\Results folder and save your file as path.py**.** The extension .py indicates the file is a Python script.

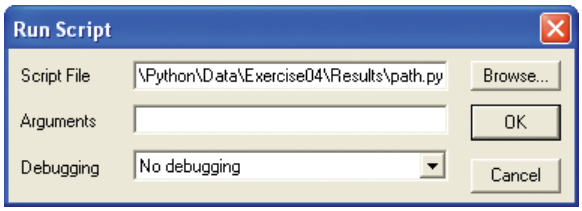
Note: Using the Save button in PythonWin is the same as using File > Save from the menu bar.

6 With your cursor still placed anywhere within the code in the path.py script window, click the Run button on the PythonWin Standard toolbar.

>>> TIP

To run a script, you can also click File > Run on the menu bar or press CTRL+R.

7 This brings up the Run Script dialog box. For now, leave the default settings and click OK.



The result `final` is printed to the Interactive Window. Notice that the syntax for Python code in the script window is the same as in the interactive interpreter. The main difference is that the script window allows you to write and save scripts without the code being run. In a typical workflow, you may use both the interactive interpreter, such as the Python window in ArcGIS or the Interactive Window in PythonWin, and the script window in a Python editor, such as PythonWin. Later exercises show examples of using both in a single workflow.

8 Close the path.py script and leave PythonWin open.

Write conditional statements

The scripts you have worked with so far use a sequential flow. In many cases, you'll want to selectively run certain portions of your code instead. That's where branching and looping statements come in.

1 On the PythonWin Standard toolbar, click the New button and confirm that you want a new script. Then save as branching.py **to the Results folder for exercise 4.**

2 Write the following code to generate a random number between 1 and 6:

```
import random
p = random.randint(1, 6)
print p
```

3 Run the script to confirm that it works correctly.

Next, you will add an `if` structure to run code based on the value of *p*.

4 Replace the line `print p` with the following:

```
if p == 6:
```

The code `p == 6` is an example of a condition—the answer is either `True` or `False`. If the answer is true, the code following the `if` statement runs. If the answer is false, there is no code left to run, and the script simply ends.

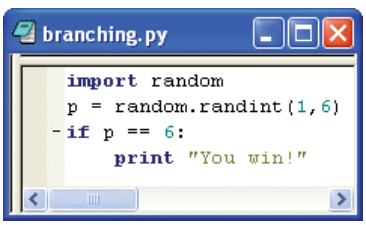
A few things to remember about the `if` structure: First, the `if` statement ends with a colon (:). Second, the lines following the `if` statement are indented. When you indent a line, the code becomes a *block*. A block consists of one or more consecutive lines of code that have the same indentation.

PythonWin assists with automatic indentation. When you press ENTER following the `if` statement colon, PythonWin automatically indents the next line of code.

5 Write the following line of code following the `if` statement:

```
print "You win!"
```

Your script should now look like the example in the figure.



6 Run the script. Running the script may result in a print statement in the Interactive Window, or nothing at all, depending on your value for *p*.

Note: *Indentation is required in Python. You can use tabs or spaces to create indentation—the style you pick is partly a matter of preference, but you should be consistent. Using either two spaces or four spaces is most common. By default, Python uses four spaces for indentation and also converts a tab to four spaces.*

Notice that the `if` structure, in this case, is not followed by anything else. If you are familiar with other programming languages, you may have expected something to follow, such as "else" or "end". In Python, the `if` structure can be used on its own or expanded by follow-up statements.

7 **In the branching.py script, place your pointer at the end of the line of code that reads `print "You win!"` and press ENTER.** Notice that the next line of code is automatically indented under the assumption you are continuing your block of code. However, in this case, you want to continue with an `else` statement, and the indentation needs to be removed.

8 **Press BACKSPACE to remove the indentation.**

9 **For the next lines of code, enter the following:**

```
else:
    print "You lose!"
```

Notice again the automatic indentation following the `else` statement. Now, your code is ready to handle both a true and a false condition.

>>> TIP

Correct indentation is key here. In this example, the `if` and the `else` statements should line up, and the two `print` statements should also line up.

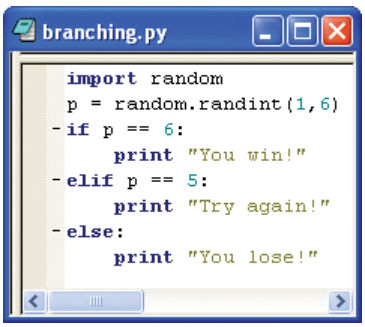
10 **Save and run the script.** By using the `if-else` structure, you account for all possible outcomes and the script prints a value to the screen every time you run it, not just when the `if` statement is `True`.

One more variant on this is the `if-elif-else` structure, which you'll use next.

11 **Insert a line above the else statement and enter the following code:**

```
elif p == 5:
    print "Try again!"
```

Your code should now look like the example in the figure.



12 Run the script a few times until the results include all three conditions. The `elif` statement is evaluated only if the `if` statement is `False`. You can use `elif` multiple times, so in principle you could specify an action for every unique possible value of the variable *p*. Like the `if` statement, the `elif` statement does not need an `else` statement to follow. You do, however, need to start this type of branching structure with an `if` statement—that is, you can't use `elif` or `else` without first using an `if` statement. Also notice that all three statements end with a colon (:) and that there is no "end" statement as there is in some programming languages.

13 Save your branching.py script and close it.

Use loop structures

There are other structures to control workflow, including the `while` loop and `for` loop structures.

1 On the PythonWin Standard toolbar, click the New button and confirm that you want a new script. Then save as whileloop.py **to the Results folder for exercise 4.**

2 Write the following code:

```
i = 0
while i <= 10:
    print i
    i += 1
```

Note: *The syntax uses the plus-equal symbol (+=), which adds a specified amount to the input value. This could also be written as* `i = i + 1`.

3 Run the script. The result is a print of the numbers 0 to 10. With each iteration over the `while` loop, the value of the variable `i` is increased by 1. The variable `i` is referred to as a *counter*. The `while` loop keeps going until the condition becomes false—that is, when the counter reaches the value of 11.

The `while` loop structure uses a syntax similar to the `if` structure: the `while` statement ends with a colon (:), and the next line of code is indented to create a block.

4 Save and close your whileloop.py script.

Next, you can try a `for` loop.

5 Create a new Python script and save as forloop.py **to the Results folder for exercise 4.**

6 Write the following code:

```
numbers = [1, 2, 3, 4, 5]
for number in numbers:
    print number
```

7 Run the script. The block of code is run for each element in the list.

8 Save and close your forloop.py script.

Looping, or iterating, over a range of numbers is a common task, and Python has a built-in function to create ranges. The `range` function has three arguments: `start`, `stop`, and `step`. The `range` function generates a list of integers, beginning with `start` and up to, but not including, `stop`, using an increment of size `step`. For example, the next code prints the numbers 0 to 100, with a step of 10.

9 Create a new Python script and save as range.py **to the Results folder for exercise 4.**

10 Write the following code:

```
for number in range(0, 101, 10):
    print number
```

11 Run the script. Iterating using a `for` loop is much more compact than using the `while` loop.

12 Save and close your range.py script.

Usually, iterating over a loop simply runs a block of code until it has used up all the sequence elements. Sometimes, however, you may want to interrupt a loop to start a new iteration or to end the loop. You can use the `break` statement to accomplish this, which you'll do next.

13 Create a new Python script and save as breakloop.py **to the Results folder for exercise 4.**

14 Write the following code:

```
from math import sqrt
for i in range(1001, 0, -1):
    root = sqrt(i)
    if root == int(root):
        print i
        break
```

15 Run the script. The code determines the largest square below 1,000. A range of integers is created, starting at 1,000 and counting down to zero (0). The negative step is used to iterate downward. When the square root of the integer is identical to the integer of the square root, you have the solution, and there is no need to continue. The solution is printed, and the loop ends.

Comment scripts

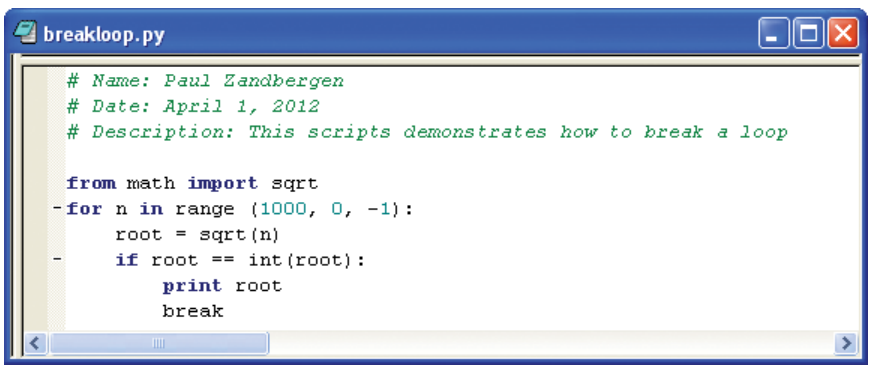
Well-developed scripts include comments that provide documentation about the script. Typically, the first few lines of a script consist of comments, but comments also occur throughout a script to explain how the script works. Comments are not executed when the script is run. In Python, a comment is preceded by the number sign (#). Any text that comes after the number sign is ignored during the execution of the script.

Next, you will add some comments that could prove useful in almost any script you write.

1 In the breakloop.py script, place your pointer at the very beginning of the code and press ENTER. At the top of the script, type the following code:

```
# Name: <your name>
# Date: <current date>
# Description: This script demonstrates how to break a loop
```

Your script window should now look like the example in the figure. Notice that the PythonWin editor recognizes comments and shows them in green italics.



Adding a comment just before a particular line or block of code can help other users understand it, as well as serve as a personal reminder about the code's meaning.

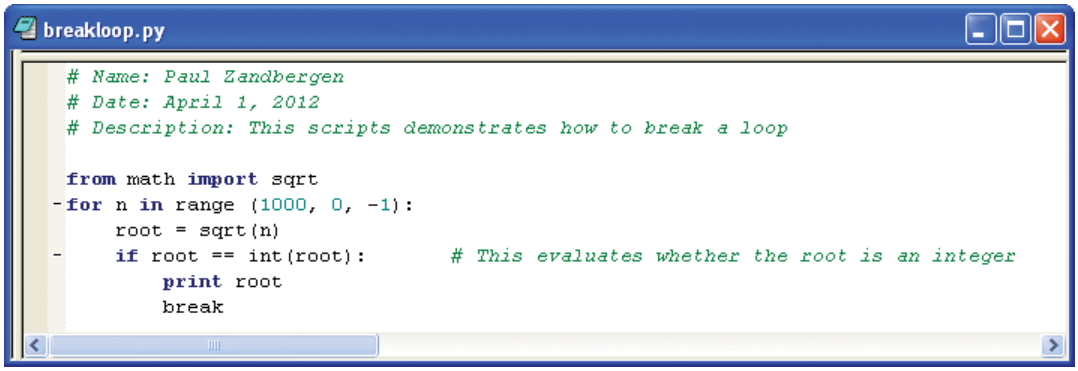
>>> TIP

Adding a line of space in your code is optional and has no effect on running the script. Typically, lines of space are added for readability. For example, it is common to add a line before or after comments or to keep lines of related code separate from other sections. This becomes more important as your scripts get longer.

2 After the `if root == int(root)` code, enter a few tabs and then the code:

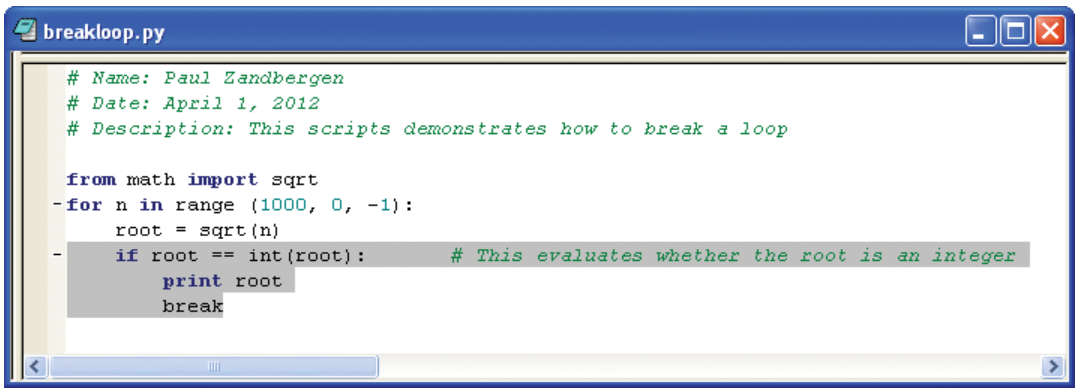
```
    # This evaluates when the root is an integer
```

Inserting comments allows you to enter specific comments to explain very specific parts of your code.

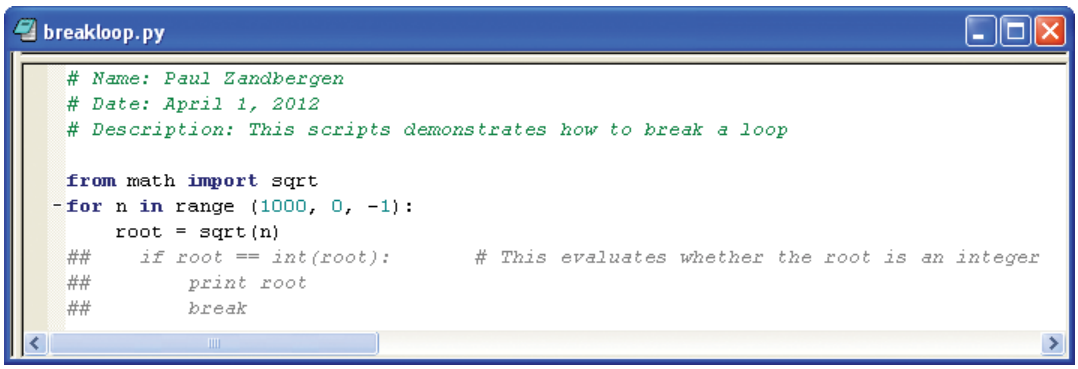


A related technique is *commenting out* several lines of code all at once. Say, for example, you have written some code and you've tested it. Now you want to try another approach without having to delete the code you already have.

3 In the breakloop.py script, highlight the last three lines of code.



4 Right-click the highlighted section of code and click Source code > Comment out region. Notice that this places double number signs (##) in front of the lines of code (you could do this manually as well, but it is much faster to comment them out all at one time). Now these lines of code are skipped when the script is run. To make the code active again, highlight the commented lines, right-click, and click Source code > Uncomment region.



5 Save your script and close PythonWin.

Check for errors

It is relatively easy to make small mistakes in your Python code as a result of spelling and other syntax errors. Next, you can take a look at some simple ways to identify and correct errors. More advanced techniques are covered in chapter 11.

Start with some simple syntax errors.

1 Start ArcMap and open the Python window. Run the following code, in which "print" is intentionally misspelled:

```
>>> pint "Hello World!"
```

There is a typo in the print statement, and the result is a syntax error:

```
Parsing error SyntaxError: invalid syntax
(line 1)
```

Notice that the error message indicates both the type of error (parsing error) and where it occurred (line 1). You can try to minimize typos by using the prompts provided as you start typing.

2 Try a small variation of your print statement by running the following code:

```
>>> print "Hello World!
```

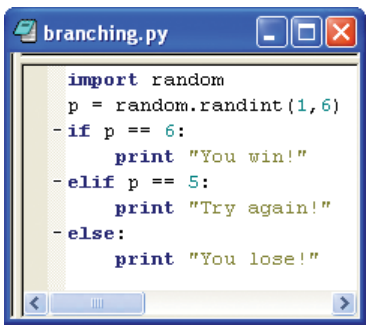
There are no closing quotation marks at the end of the line of code, resulting in a syntax error:

```
Parsing error SyntaxError: EOL while scanning
string literal (line 1)
```

Notice that the error message provides specific details on the nature of the error. EOL stands for End of Line, so you know where to look to correct the error. There are a lot of different types of error messages—too many to worry about at this point, but the basic idea is that error messages provide information on both the nature of the error and where the error occurs. Next, you will look at how this works for scripts consisting of multiple lines of code.

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

4 Start PythonWin and open your branching.py script from earlier in the exercise. It should look like the example in the figure.



Next, try adding a small error.

5 Place your pointer at the end of the third line of code and remove the colon (:) at the end of the `if` statement:

```
if p == 6
```

6 Place your pointer anywhere in the first line of code and click the Check button . This checks the syntax of the code without running it. Notice that the cursor is placed at the end of the third line of code where the syntax error is located, and the status bar at the bottom of the PythonWin interface reads, "Failed to check – syntax error – invalid syntax." The Check option provides a quick way to test the syntax prior to running your script.

7 Now try running the script. In this case, running the script returns the same result: the cursor is placed at the end of the third line of code, and the status bar at the bottom indicates a syntax error. For more complicated scripts, it is useful to first identify and remove syntax errors using the Check option, and then run the script to see if there are other errors.

8 Correct the syntax error by placing a colon (:) at the end of the `if` statement:

```
if p == 6:
```

9 Now introduce a different error by placing a typo in the `randint` function:

```
p = random.randinr(0, 6)
```

10 Check the syntax of your script by clicking the Check button. Clicking Check confirms that there are no syntax errors in your script. Syntax checking examines the statements and expressions in your code but does not check for other errors, such as naming a function incorrectly. Only when you run the code will you discover that the function `randinr` does not exist.

11 Try running your script. Notice a fairly lengthy error message that is printed to the Interactive Window. The last three lines read as follows:

```
File "......\branching.py", line 2, in <module>
p = random.randinr(0,6)
AttributeError: 'module' object has no attribute 'randinr'
```

The error message provides a clear indication of where the error is located (line 2 of the code in which you are working with a module) and what the error is (`randinr` is not a function of this module).

12 Close PythonWin. There is no need to save the changes to your script.

Challenge exercises

Challenge 1

Create a script that examines a string for the occurrence of a particular letter, as you did previously in this exercise for "Geographic Information Systems." If the letter occurs in the text (for example, the letter Z), the string "Yes" should be printed to the Interactive Window. If the letter does not occur in the text, the string "No" should be printed.

Challenge 2

Create a script that examines a list of numbers without duplicates (for example, 2, 8, 64, 16, 32, 4) and determines the second-largest number.

Challenge 3

Create a script that examines a list of numbers (for example, 2, 8, 64, 16, 32, 4, 16, 8) to determine whether it contains duplicates. The script should print a meaningful result, such as "The list provided contains duplicate values" or "The list provided does not contain duplicate values."

An optional addition is to remove the duplicates from the list.

Challenge 4

Consider the following list:

```
mylist = ["Athens", "Barcelona", "Cairo", "Florence", "Helsinki"]
```

Determine the results of the following:

a) `len(mylist)`
b) `mylist[2]`
c) `mylist[1:]`
d) `mylist[-1]`
e) `mylist.index("Cairo")`
f) `mylist.pop(1)`
g) `mylist.sort(reverse = True)`
h) `mylist.append("Berlin")`

These operations are all to be performed on the original list—that is, not as a sequence of operations. Try to determine the answer manually first, and then check your result by running the code.

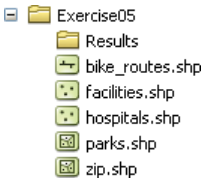
Exercise 5
Geoprocessing using Python

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.**

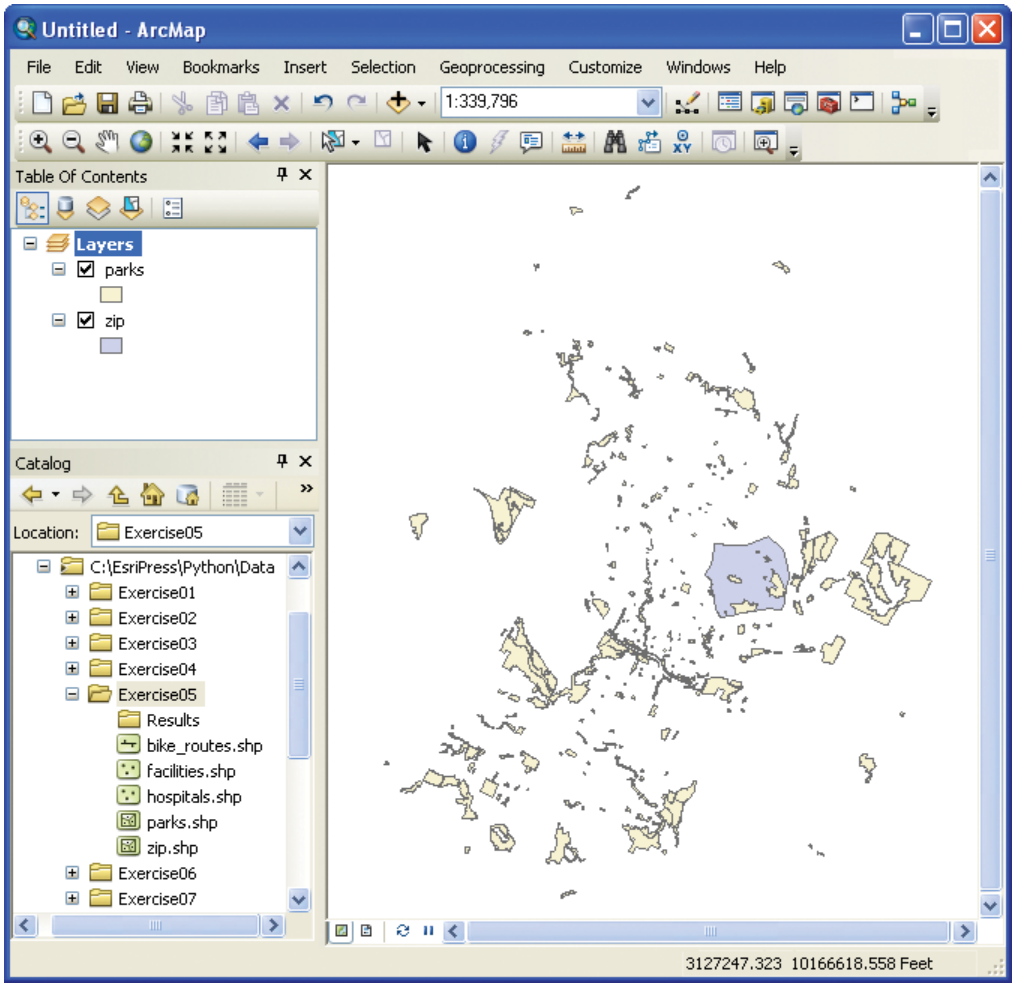
2 **Browse to the exercise 5 folder.**



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

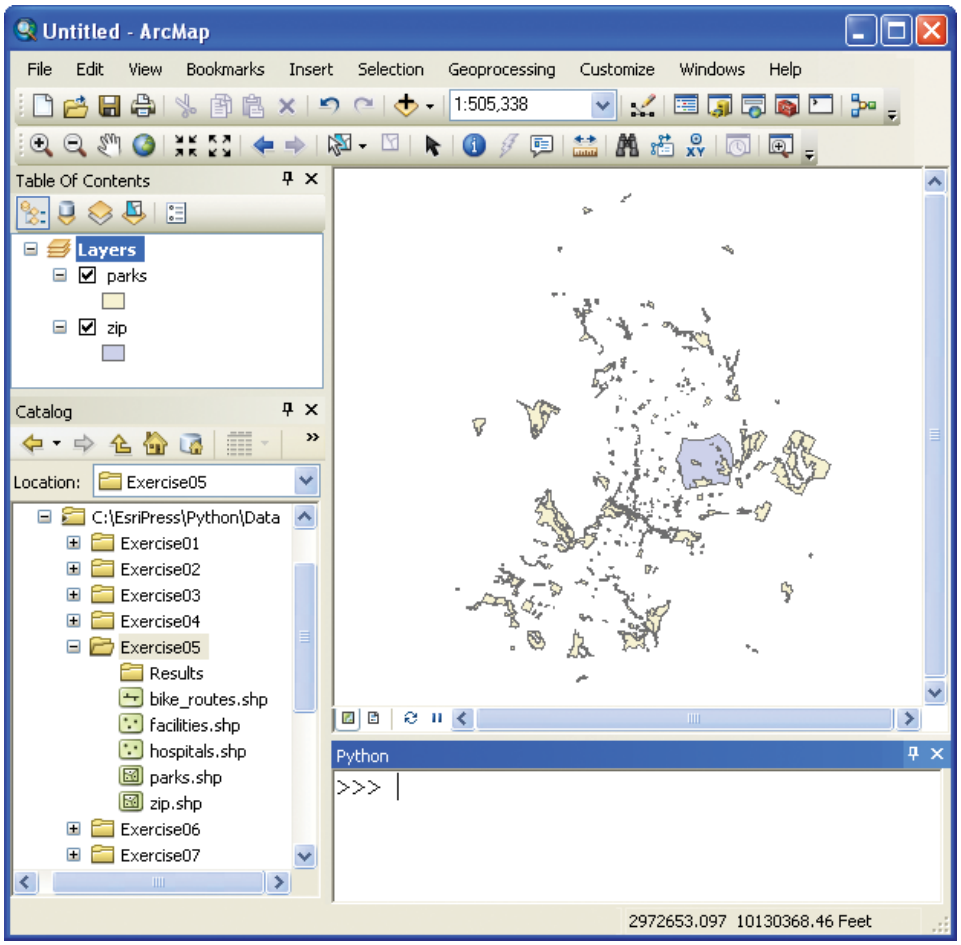
3 **Drag the parks.shp and zip.shp files to the ArcMap Table Of Contents window.**

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.

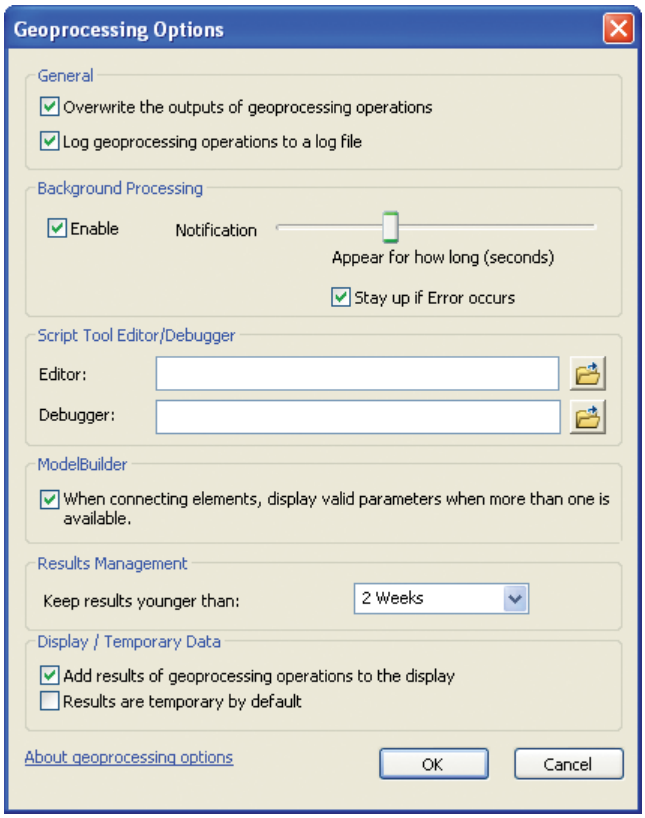


Note: *Docking the Catalog and Python windows is not required, but it can help to organize your available desktop space. Typically, it is helpful if the Python window is fairly wide to make it easier to see longer lines of code.*

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.

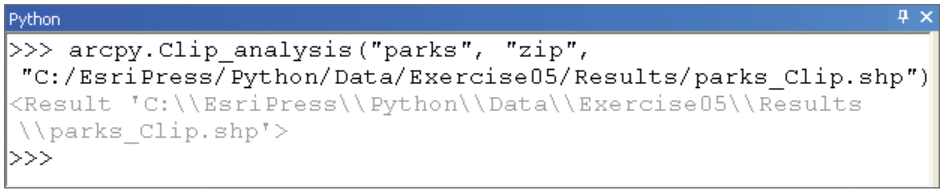
10 In the Python window, at the command prompt, enter the following code and press ENTER:

```
>>> 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.



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.

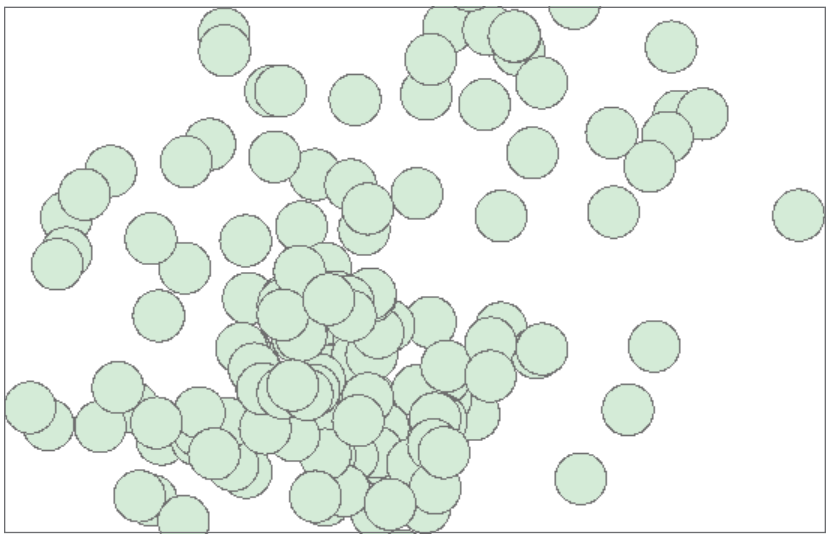
Next, you will use another geoprocessing tool.

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 ("", "").

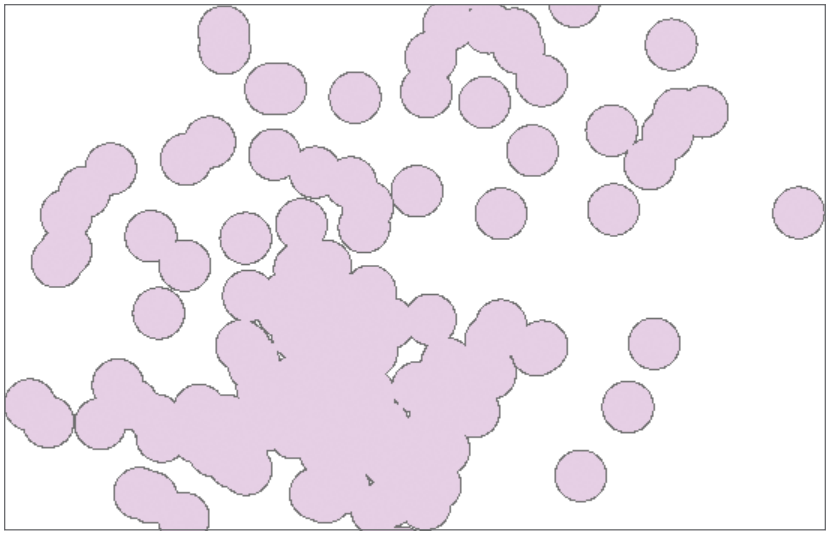
14 At the prompt, use the UP ARROW key to bring up the previous line of code:

```
>>> 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

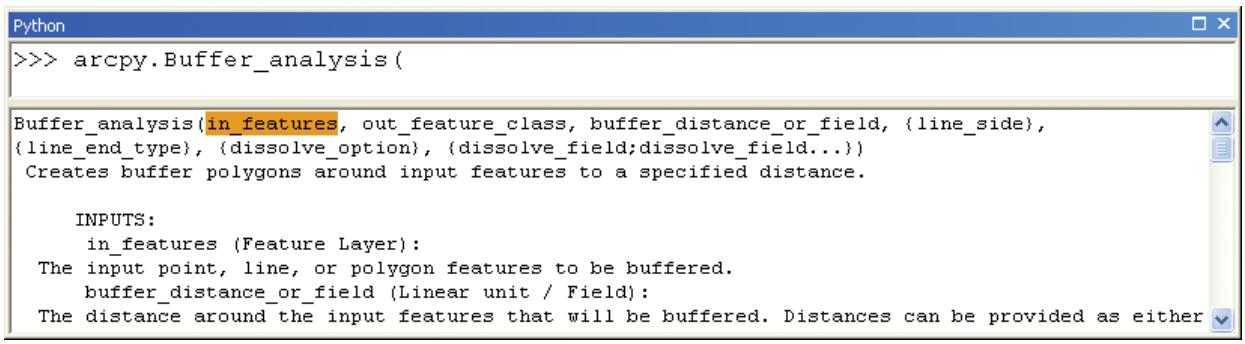
Working with geoprocessing tools in Python requires a good understanding of the syntax of the tools. The proper syntax can be reviewed in a number of ways.

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(
```

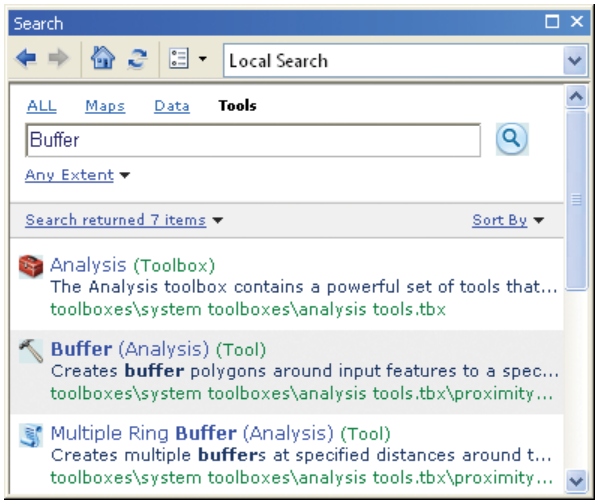
Entering the code brings up the syntax for the tool.



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.

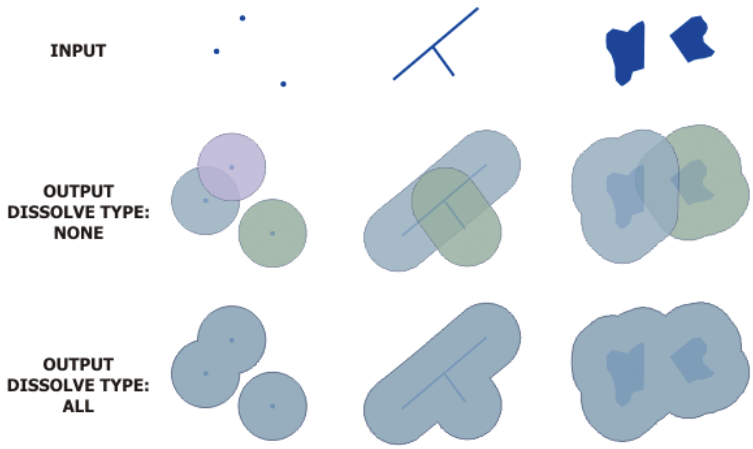
Buffer (Analysis)

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:

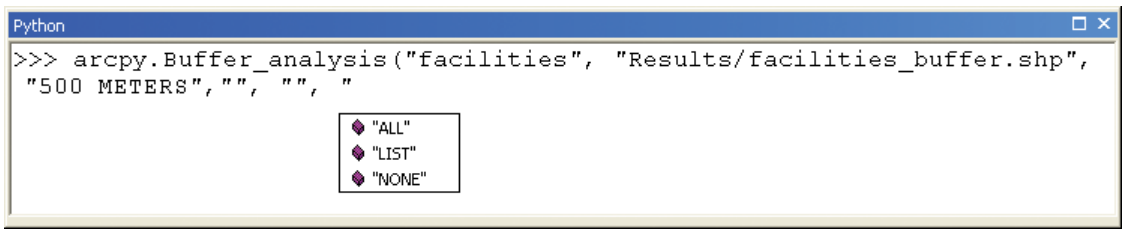
```
# Name: Buffer.py
# Description: Find areas of suitable vegetation which exclude areas heavily impacted by major roads

# 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"`.



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:

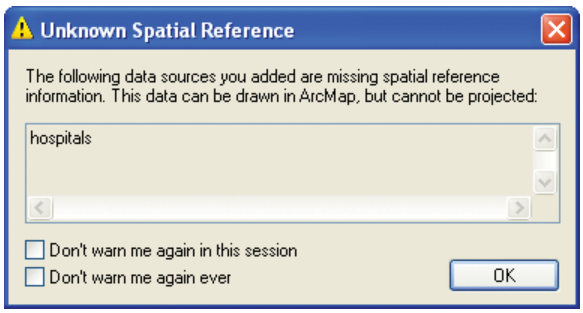
```
>>> arcpy.Usage("Exists")
```

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`.

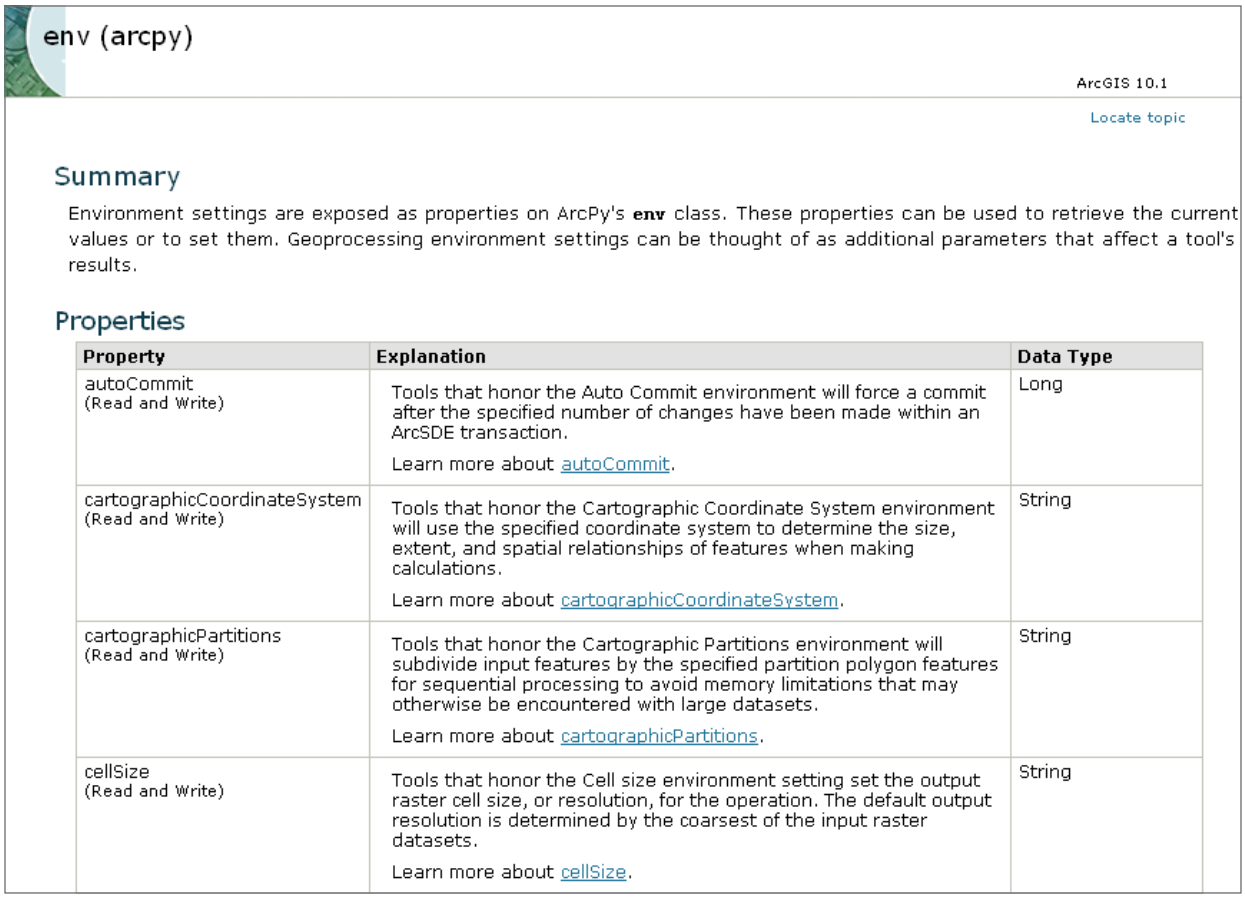
```
>>> print spatial_ref.XYResolution
```

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.

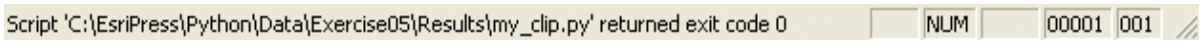
2 On the PythonWin Standard toolbar, click the New button. Select Python Script and click OK.

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()
```

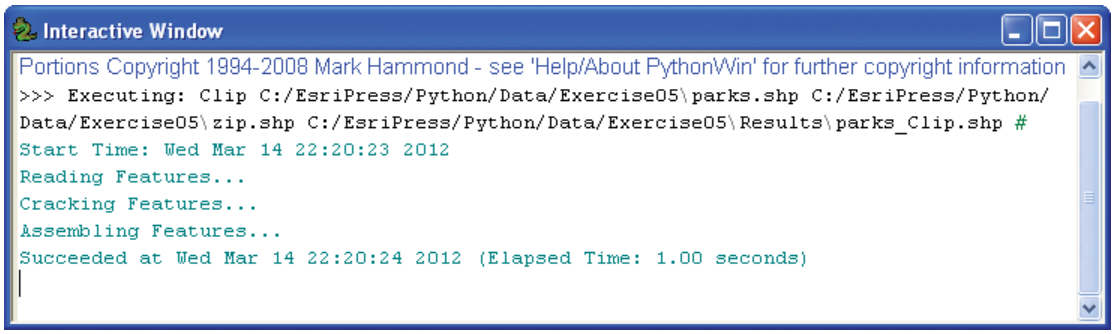
4 On the PythonWin Standard toolbar, click the Save button and save as my_clip.py **to the C:\EsriPress\Python\Data\Exercise05\Results folder.**

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.



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.

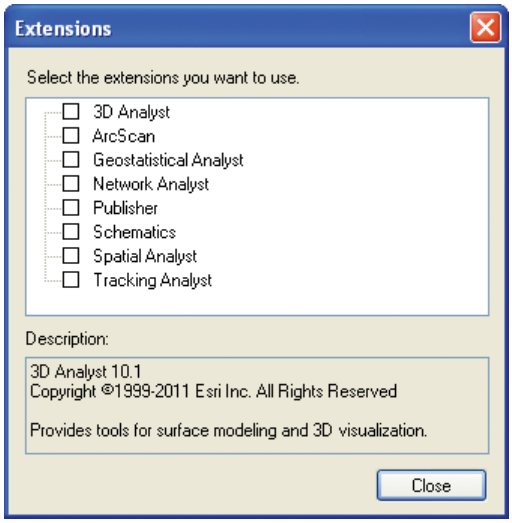
3 At the command prompt, enter and run the following code:

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

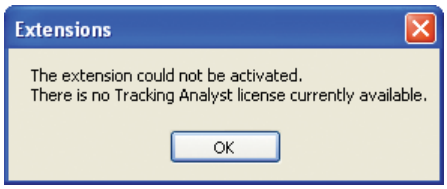
If you are running ArcGIS for Desktop Advanced (`arcinfo`), the code returns `AlreadyInitialized` or `Available`. If you are running a lower license level, it returns `Unavailable`.

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. ➔

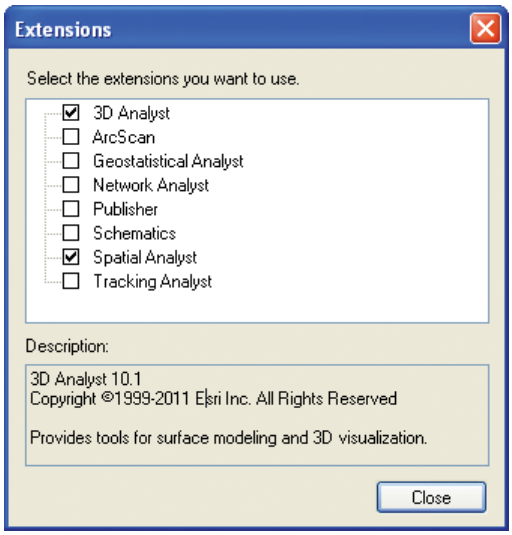


5 Select the check boxes for the extensions on the list to see which ones have licenses available. You may get a warning message for some of them, like the example in the figure.



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 different, 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.

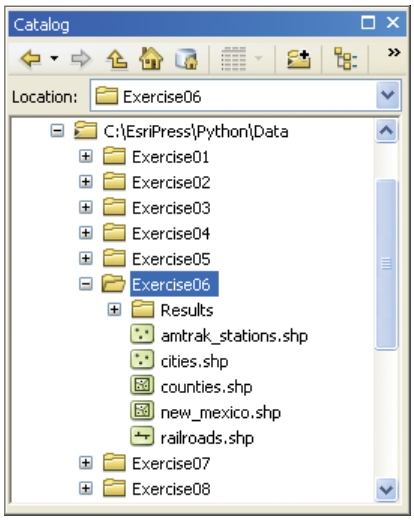
Challenge 4

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: ..."`.

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.

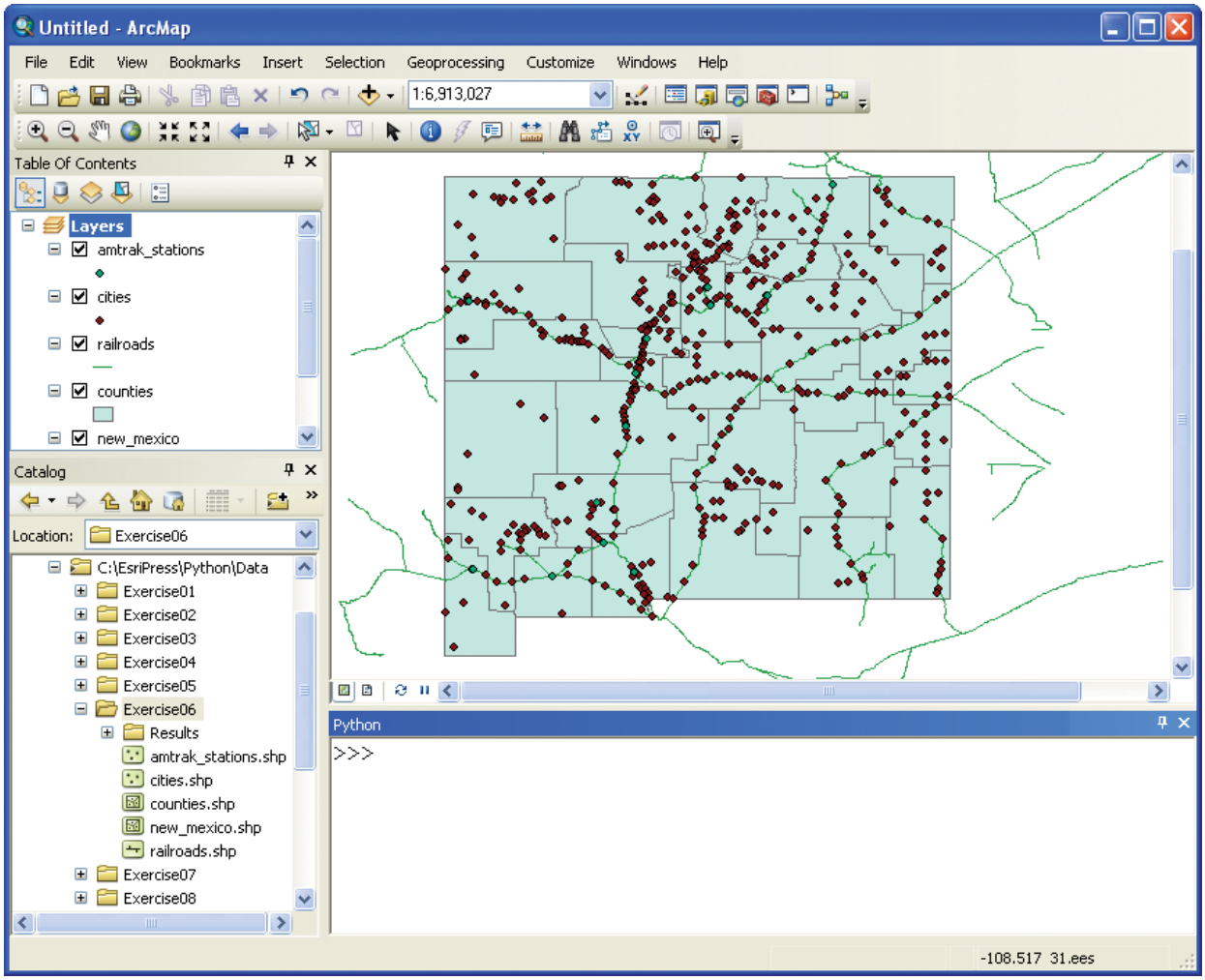
10 **Close PythonWin.**

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. ➔

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.

Exercise 7
Manipulating spatial data

Work with search cursors

Cursors are used to iterate over the rows in a table. Cursor functions create cursor objects, which can be used to access the row objects. Several methods exist to manipulate these row objects.

Search cursors are used to search records and to carry out SQL expressions in Python.

1 **Start ArcMap and open a new blank map document. Open the Catalog window. Navigate to the Exercise07 folder and drag the shapefile airports.shp into the map document.**

2 **In the ArcMap table of contents, right-click the airports layer and click Open Attribute Table. Review the fields of the airports.** Notice that there is a field called NAME and that the table contains 221 records.

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

4 **Start PythonWin. Create a new Python script and save as** printvalues.py **to your C:\EsriPress\Python\Data\Exercise07\Results folder.**

5 **Enter the following code:**

```
import arcpy
from arcpy import env
env.workspace = "C:/EsriPress/Python/Data/Exercise07"
fc = "airports.shp"
cursor = arcpy.da.SearchCursor(fc, ["NAME"])
for row in cursor:
    print "Airport name = {0}".format(row[0])
```

Notice that the script creates a search cursor on the feature class and uses a `for` loop to iterate over all the rows of the attribute table.

6 Save and run the script.

The result is a list of the names of all the airports, as follows:

```
Airport name = Hyder
Airport name = Chignik Lagoon
Airport name = Koyuk
Airport name = Kivalina
Airport name = Ketchikan Harbor
Airport name = Metlakatla
Airport name = Waterfall
 ...
```

Note: *Be careful printing results to the Interactive Window because a feature class or table could contain millions of records. In the preceding example, it was confirmed in advance that the number of records was relatively limited.*

Use search cursors with SQL in Python

Search cursors can be used to carry out SQL expressions in Python.

1 In PythonWin, create a new Python script and save as SQL.py **to the Results folder for exercise 7.**

2 Enter the following code:

```
import arcpy
from arcpy import env
env.workspace = "C:/EsriPress/Python/Data/Exercise07"
fc = "airports.shp"
cursor = arcpy.da.SearchCursor(fc, ["NAME"], '"TOT_ENP" > 100000')
for row in cursor:
    print row[0]
del row
del cursor
```

3 Save and run the script. The result is a list of the names of the airports for which the SQL expression is true. The field TOT_ENP is a measure of the number of passengers. The list is as follows:

```
Ketchikan
Juneau International
Kenai Municipal
Fairbanks International
Bethel
Ted Stevens Anchorage International
```

Take another look at the SQL expression used: `"TOT_ENP" > 100000`. Field delimiters for shapefiles consist of double quotation marks—for example, `"TOT_ENP"`—but there are no quotation marks around the value of `100000` because TOT_ENP is a numeric field. The entire SQL expression needs to be in quotation marks, because the `WHERE` clause in the syntax of the search cursor is a string. This results in the SQL expression, `'"TOT_ENP" > 100000'`.

This syntax can create complications. For example, for text fields in SQL expressions, the values require single quotation marks—for example, `"NAME" = 'Ketchikan'`. The statement in the `WHERE` clause needs to be in quotation marks, but whether you use double quotation marks (" ") or single quotation marks (' '), the statement will produce a syntax error. The solution is to use the escape character (\), which would otherwise cause a syntax error, in front of the quotation marks. In Python, a backslash within a string is interpreted as an escape character, which is a signal that the next character is to be given a special interpretation. So instead of `'"NAME" = 'Ketchikan''`, the expression becomes `'"NAME" = \'Ketchikan\''`.

4 Modify the SQL expression in the script as follows:

```
cursor = arcpy.da.SearchCursor(fc, ["NAME"], '"FEATURE" = ➔
➔ \'Seaplane Base\'')
```

5 Save and run the script.

The result is a list of the names of the airports for which the SQL expression is true, as follows:

```
Hyder
Ketchikan Harbor
Metlakatla
Waterfall
Kasaan
Hollis
Craig
...
```

There are other complications when working with SQL expressions. Specifically, the field delimiters vary with the format of the feature class, as follows:

- Shapefiles and file geodatabase feature classes use double quotation marks (" ")—for example, `"NAME"`.
- Personal geodatabase feature classes use square brackets ([ ])—for example, [NAME].
- ArcSDE geodatabase feature classes do not use any delimiters—for example, NAME.

When a tool like Select By Attributes or other dialog-driven queries is used, this syntax is automatically applied, but in scripting, this can be handled using the `AddFieldDelimiters` function.

6 Modify the script as follows:

```
import arcpy
from arcpy import env
env.workspace = "C:/EsriPress/Python/Data/Exercise07"
fc = "airports.shp"
delimitedField = arcpy.AddFieldDelimiters(fc, "COUNTY")
cursor = arcpy.da.SearchCursor(fc, ["NAME"], delimitedField + " =
'Anchorage Borough'")
for row in cursor:
    print row[0]
del row
del cursor
```

7 Save and run the script.

The result is a list of the names of the airports for which the SQL expression is true, as follows:

```
Girdwood
Merrill Field
Lake Hood
Elmendorf Air Force Base
Ted Stevens Anchorage International
```

SQL is also used in a number of geoprocessing tools, and a similar approach can be used to create valid SQL expressions, in general.

8 Save the existing SQL.py script as Select.py **to the Exercise07 folder.**

9 Modify the script as follows:

```
import arcpy
from arcpy import env
env.workspace = "C:/EsriPress/Python/Data/Exercise07"
infc = "airports.shp"
outfc = "Results/airports_anchorage.shp"
delimitedfield = arcpy.AddFieldDelimiters(infc, "COUNTY")
arcpy.Select_analysis(infc, outfc, delimitedfield + " = 
 'Anchorage Borough'")
```

10 Save and run the script.

11 Start ArcMap and open the Catalog window.

12 In the Catalog window, navigate to the Results folder for exercise 7 and confirm that the new shapefile was created with five point features.

Work with update cursors

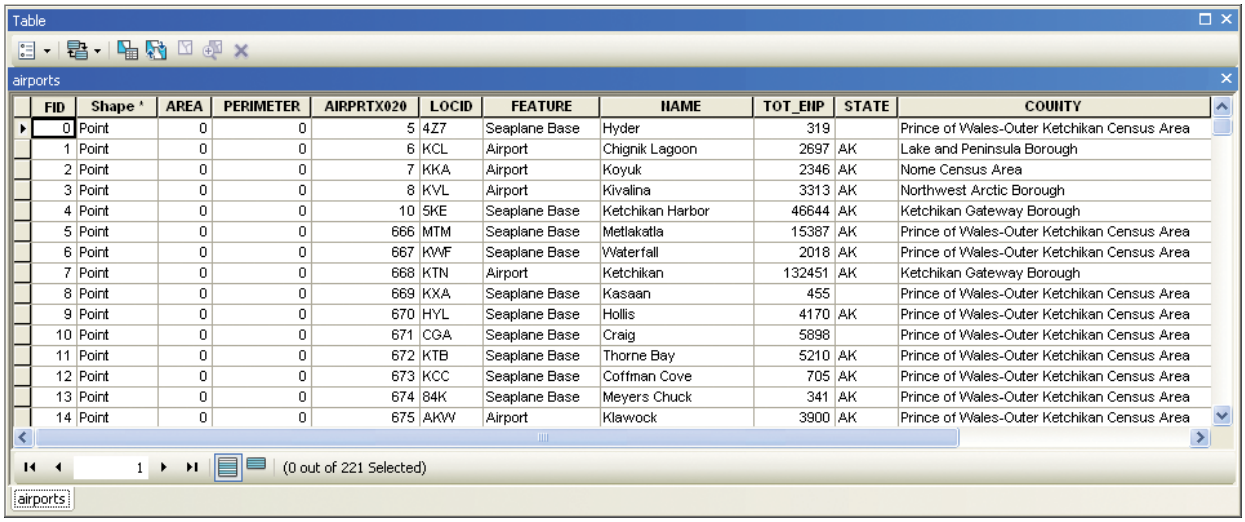
Two other cursor types can be used to work with row objects. Update cursors are used to make changes to existing records, and insert cursors are used to add new records. First, you will use an update cursor to update attribute values and delete records. Because this will permanently modify the data, it is a good idea to copy the data first.

1 In the Catalog window, navigate to the Exercise07 folder.

2 Drag the shapefile airports.shp into the map document.

3 In the ArcMap table of contents, right-click the airports layer and click Open Attribute Table.

4 Scroll over to the field STATE. Notice that some of the values in this field are blank.



Table

airports

FID	Shape *	AREA	PERIMETER	AIRPRTX020	LOCID	FEATURE	NAME	TOT_ENP	STATE	COUNTY
0	Point	0	0	5	4Z7	Seaplane Base	Hyder	319		Prince of Wales-Outer Ketchikan Census Area
1	Point	0	0	6	KCL	Airport	Chignik Lagoon	2697	AK	Lake and Peninsula Borough
2	Point	0	0	7	KKA	Airport	Koyuk	2346	AK	Nome Census Area
3	Point	0	0	8	KVL	Airport	Kivalina	3313	AK	Northwest Arctic Borough
4	Point	0	0	10	5KE	Seaplane Base	Ketchikan Harbor	46644	AK	Ketchikan Gateway Borough
5	Point	0	0	666	MTM	Seaplane Base	Metlakatla	15387	AK	Prince of Wales-Outer Ketchikan Census Area
6	Point	0	0	667	KWF	Seaplane Base	Waterfall	2018	AK	Prince of Wales-Outer Ketchikan Census Area
7	Point	0	0	668	KTN	Airport	Ketchikan	132451	AK	Ketchikan Gateway Borough
8	Point	0	0	669	KXA	Seaplane Base	Kasaan	455		Prince of Wales-Outer Ketchikan Census Area
9	Point	0	0	670	HYL	Seaplane Base	Hollis	4170	AK	Prince of Wales-Outer Ketchikan Census Area
10	Point	0	0	671	CGA	Seaplane Base	Craig	5898		Prince of Wales-Outer Ketchikan Census Area
11	Point	0	0	672	KTB	Seaplane Base	Thorne Bay	5210	AK	Prince of Wales-Outer Ketchikan Census Area
12	Point	0	0	673	KCC	Seaplane Base	Coffman Cove	705	AK	Prince of Wales-Outer Ketchikan Census Area
13	Point	0	0	674	84K	Seaplane Base	Meyers Chuck	341	AK	Prince of Wales-Outer Ketchikan Census Area
14	Point	0	0	675	AKW	Airport	Klawock	3900	AK	Prince of Wales-Outer Ketchikan Census Area

(0 out of 221 Selected)

airports

5 Close the attribute table.

6 In the Catalog window, navigate to the Exercise07 folder and copy the airports.shp feature class to the Results folder so you can make edits without having to worry about keeping the original intact.

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

Note: Working with insert and update cursors is just like editing, and having the feature class you want to work with in a script be open at the same time in ArcMap may result in errors because of a shared lock ArcMap places on the feature class.

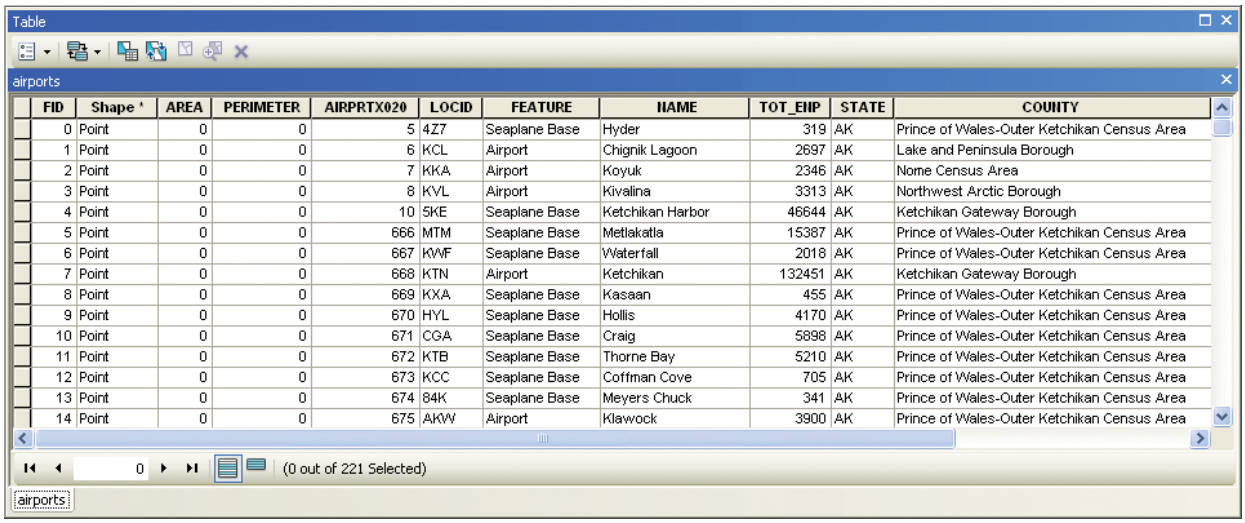
8 In PythonWin, create a new Python script and save as Update.py **to the Results folder for exercise 7.**

9 Enter the following code:

```
import arcpy
from arcpy import env
env.workspace = "C:/EsriPress/Python/Data/Exercise07"
fc = "Results/airports.shp"
delimfield = arcpy.AddFieldDelimiters(fc, "STATE")
cursor = arcpy.da.UpdateCursor(fc, ["STATE"], delimfield + " <> 'AK'")
for row in cursor:
    row[0] = "AK"
    cursor.updateRow(row)
del row
del cursor
```

10 Save and run the script.

11 Start ArcMap, open the Catalog window, navigate to the Exercise07 folder, and drag the airports.shp file into the data frame. Confirm that the values in the STATE field have been updated.



FID	Shape *	AREA	PERIMETER	AIRPRTX020	LOCID	FEATURE	NAME	TOT_ENP	STATE	COUNTY
0	Point	0	0	5	4Z7	Seaplane Base	Hyder	319	AK	Prince of Wales-Outer Ketchikan Census Area
1	Point	0	0	6	KCL	Airport	Chignik Lagoon	2697	AK	Lake and Peninsula Borough
2	Point	0	0	7	KKA	Airport	Koyuk	2346	AK	Nome Census Area
3	Point	0	0	8	KVL	Airport	Kivalina	3313	AK	Northwest Arctic Borough
4	Point	0	0	10	5KE	Seaplane Base	Ketchikan Harbor	46644	AK	Ketchikan Gateway Borough
5	Point	0	0	666	MTM	Seaplane Base	Metlakatla	15387	AK	Prince of Wales-Outer Ketchikan Census Area
6	Point	0	0	667	KWF	Seaplane Base	Waterfall	2018	AK	Prince of Wales-Outer Ketchikan Census Area
7	Point	0	0	668	KTN	Airport	Ketchikan	132451	AK	Ketchikan Gateway Borough
8	Point	0	0	669	KXA	Seaplane Base	Kasaan	455	AK	Prince of Wales-Outer Ketchikan Census Area
9	Point	0	0	670	HYL	Seaplane Base	Hollis	4170	AK	Prince of Wales-Outer Ketchikan Census Area
10	Point	0	0	671	CGA	Seaplane Base	Craig	5898	AK	Prince of Wales-Outer Ketchikan Census Area
11	Point	0	0	672	KTB	Seaplane Base	Thorne Bay	5210	AK	Prince of Wales-Outer Ketchikan Census Area
12	Point	0	0	673	KCC	Seaplane Base	Coffman Cove	705	AK	Prince of Wales-Outer Ketchikan Census Area
13	Point	0	0	674	84K	Seaplane Base	Meyers Chuck	341	AK	Prince of Wales-Outer Ketchikan Census Area
14	Point	0	0	675	AKW	Airport	Klawock	3900	AK	Prince of Wales-Outer Ketchikan Census Area

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

In addition to updating attributes using the `updateRow` method, search cursors can also be used to delete records, which you'll do next.

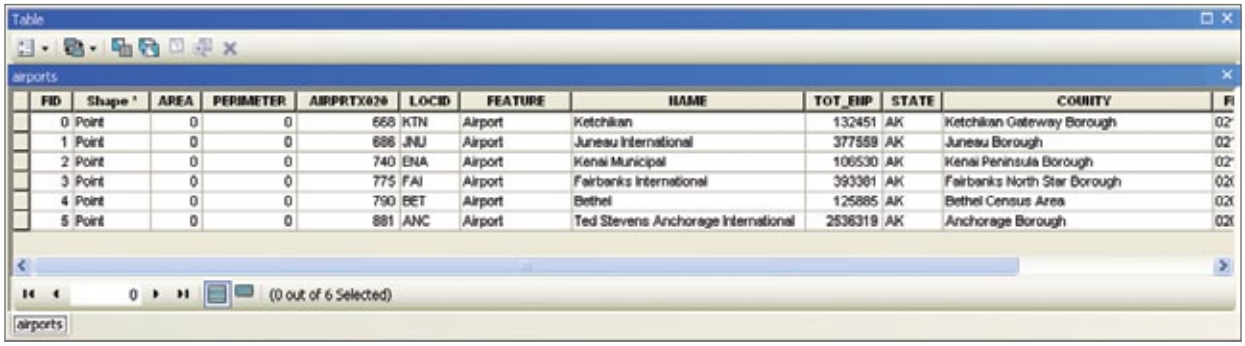
13 In PythonWin, create a new Python script and save as Delete.py **to the Results folder for exercise 7.**

14 Enter the following code:

```
import arcpy
from arcpy import env
env.workspace = "C:/EsriPress/Python/Data/Exercise07"
fc = "Results/airports.shp"
cursor = arcpy.da.UpdateCursor(fc, ["TOT_ENP"])
for row in cursor:
    if row[0] < 100000:
        cursor.deleteRow()
del row
del cursor
```

15 Save and run the script.

16 Start ArcMap, open the Catalog window, navigate to the Results folder for exercise 7, and drag the airports.shp file into the data frame. Confirm that all the records with fewer than 100,000 passengers have been deleted.



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

Work with insert cursors

Insert cursors are used to create new records.

1 In PythonWin, create a new Python script and save as insert.py **to the Results folder for exercise 7.**

2 Enter the following code:

```
import arcpy
from arcpy import env
env.workspace = "C:/EsriPress/Python/Data/Exercise07"
fc = "Results/airports.shp"
cursor = arcpy.da.InsertCursor(fc, "NAME")
cursor.insertRow(["New Airport"])
del cursor
```

3 Save and run the script.

4 Start ArcMap, open the Catalog window, navigate to the Results folder for exercise 7, and drag the airports.shp file into the data frame. Confirm that a new record has been added with the name New Airport **— the other fields are blank.**

FID	Shape *	AREA	PERIMETER	AIRPRTX020	LOCID	FEATURE	NAME	TOT_ENP	STATE	COUNTY	FI
0	Point	0	0	668	KTN	Airport	Ketchikan	132451	AK	Ketchikan Gateway Borough	02
1	Point	0	0	686	JNU	Airport	Juneau International	377559	AK	Juneau Borough	02
2	Point	0	0	740	ENA	Airport	Kenai Municipal	106530	AK	Kenai Peninsula Borough	02
3	Point	0	0	775	FAI	Airport	Fairbanks International	393381	AK	Fairbanks North Star Borough	02
4	Point	0	0	790	BET	Airport	Bethel	125885	AK	Bethel Census Area	02
5	Point	0	0	881	ANC	Airport	Ted Stevens Anchorage International	2536319	AK	Anchorage Borough	02
6	Point	0	0	0			New Airport	0			

Note: *Although a new record has been added and the attributes can be given values using the insert cursor, the new record does not have a geometry yet. This is covered in chapter 8.*

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

Validate table and field names

ArcPy contains functions to validate the names of tables and fields. This prevents attempts to create invalid names, such as those with spaces or invalid characters.

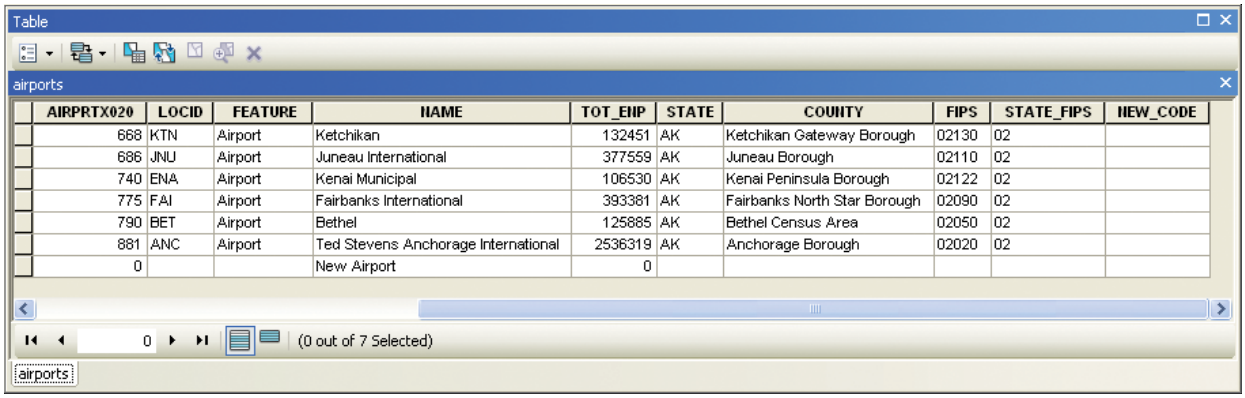
1 In PythonWin, create a new Python script and save as validatefield.py **to the Results folder for exercise 7.**

2 Enter the following code:

```
import arcpy
from arcpy import env
env.workspace = "C:/EsriPress/Python/Data/Exercise07"
fc = "Results/airports.shp"
newfield = "NEW CODE"
fieldtype = "TEXT"
fieldname = arcpy.ValidateFieldName(newfield)
arcpy.AddField_management(fc, fieldname, fieldtype, "", "", 12)
```

3 Save and run the script.

4 Start ArcMap, open the Catalog window, navigate to the Results folder for exercise 7, and drag the airports.shp file into the data frame. Confirm that a new field has been added with the name NEW_CODE**.** The space has been replaced by an underscore (_).



AIRPRTX020	LOCID	FEATURE	NAME	TOT_ENP	STATE	COUNTY	FIPS	STATE_FIPS	NEW_CODE
668	KTN	Airport	Ketchikan	132451	AK	Ketchikan Gateway Borough	02130	02	
686	JNU	Airport	Juneau International	377559	AK	Juneau Borough	02110	02	
740	ENA	Airport	Kenai Municipal	106530	AK	Kenai Peninsula Borough	02122	02	
775	FAI	Airport	Fairbanks International	393381	AK	Fairbanks North Star Borough	02090	02	
790	BET	Airport	Bethel	125885	AK	Bethel Census Area	02050	02	
881	ANC	Airport	Ted Stevens Anchorage International	2536319	AK	Anchorage Borough	02020	02	
0			New Airport	0					

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

6 In PythonWin, modify the script as follows:

```
newfield = "NEW?*&$"
```

The characters ?, *, &, and $ are all invalid as field names.

7 Save and run the script.

8 Start ArcMap, open the Catalog window, navigate to the Results folder for exercise 7, and drag the airports.shp file into the data frame. Confirm that the name of the new field has been modified to NEW____**.** Each of the invalid characters has been replaced by an underscore.

>>> TIP

Validating table and field names does not determine whether the field name already exists. This requires checking the new name against the names of the existing fields.

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

10 Modify the script as follows:

```
import arcpy
from arcpy import env
env.workspace = "C:/EsriPress/Python/Data/Exercise07"
fc = "Results/airports.shp"
newfield = "NEW CODE"
fieldtype = "TEXT"
fieldname = arcpy.ValidateFieldName(newfield)
fieldlist = arcpy.ListFields(fc)
fieldnames = []
for field in fieldlist:
    fieldnames.append(field.name)
if fieldname not in fieldnames:
    arcpy.AddField_management(fc, fieldname, fieldtype, "", "", 12)
    print "New field has been added."
else:
    print "Field name already exists."
```

11 Save and run the script.

The preceding script creates a list of field objects using the `ListFields` function. The names of these field objects are placed in a new empty list using the `append` method. The validated name of the new field is compared to this list of field names using an `if ( ) not in` statement.

The `CreateUniqueName` function can also be used to ensure a name for a new feature class or a table is unique, but it is limited to unique names in a workspace. It cannot be used to ensure a field name is unique.

12 In PythonWin, create a new Python script and save as unique_name.py **to the Results folder for exercise 7.**

13 Enter the following code:

```
import arcpy
from arcpy import env
env.workspace = "C:/EsriPress/Python/Data/Exercise07"
fc = "airports.shp"
unique_name = arcpy.CreateUniqueName("Results/buffer.shp")
arcpy.Buffer_analysis(fc, unique_name, "5000 METERS")
```

14 Save and run the script.

15 Start ArcMap, open the Catalog window, navigate to the Results folder for exercise 7, and confirm that a new feature class called buffer.shp **has been created.**

16 Return to PythonWin and run the script again.

17 In ArcMap, in the Catalog window, confirm that a new feature class called buffer0.shp **has been created.**

If you keep running the script again, the next output files will be called **buffer1.shp, buffer2.shp**, and so on. Using the `CreateUniqueName` function can prevent accidentally overwriting files.

>>> TIP

If the feature class is not showing, you may need to refresh the Catalog view by right-clicking the folder in the ArcMap table of contents and clicking Refresh.

Challenge exercises

Challenge 1

Write a script that creates a 15,000-meter buffer around features in the airports.shp feature class classified as an airport (based on the `FEATURE` field) and a 7,500-meter buffer around features classified as a seaplane base. The results should be two separate feature classes, one for each airport type.

Challenge 2

Write a script that adds a text field to the roads.shp feature class called **FERRY** and populates this field with YES and NO values, depending on the value of the FEATURE field.

Exercise 8
Working with geometries

Work with geometry objects

Working with full geometry objects can be costly in terms of time. Geometry tokens provide shortcuts to specific geometry properties of individual features in a feature class. In the next example, you will see how to work with geometry tokens in scripting.

1 **Start PythonWin. Create a new Python script and save as** geometry.py **to your C:\EsriPress\Python\Data\Exercise08\Results folder.**

2 **Enter the following code:**

```
import arcpy
from arcpy import env
env.workspace = "C:/EsriPress/Python/Data/Exercise08"
fc = "rivers.shp"
cursor = arcpy.da.SearchCursor(fc, ["SHAPE@LENGTH"])
length = 0
for row in cursor:
    length = length + row[0]
print length
```

3 **Save and run the script.** This prints the total length of all the river segments to the Interactive Window. The spatial reference object can be used to determine the units.

4 **Replace the** `print length` **statement with the following lines of code:**

```
units = arcpy.Describe(fc).spatialReference.linearUnitName
print str(length) + " " + units
```

5 **Save and run the script.**

This prints the total length of all the features, followed by the units, to the Interactive Window, as follows:

```
256937.409437 Meter
```

In the `print` statement, the length is converted to a string because if it were a number followed by a plus sign (+), it would suggest a calculation.

Read geometries

In addition to the geometry properties of a feature, a feature's individual vertices can also be accessed. You'll do that next.

1 In PythonWin, create a new Python script and save as points.py **to the Results folder for exercise 8.**

2 Enter the following code:

```
import arcpy
from arcpy import env
env.workspace = "C:/EsriPress/Python/Data/Exercise08"
fc = "dams.shp"
cursor = arcpy.da.SearchCursor(fc, ["SHAPE@XY"])
for row in cursor:
    x, y = row[0]
    print("{0}, {1}".format(x, y))
```

3 Save and run the script.

This prints a pair of x,y coordinates for each point, as follows:

```
852911.336075 2220578.93538
792026.89767 2310863.76822
784830.427658 2315171.53882
741687.458878 2321601.76549
702480.327773 2340545.89511
623387.057118 2361903.92116
...
```

Working with other geometry types, such as polylines and polygons, requires some extra code because an array of point objects is returned for each feature. As a result, an extra iteration is required to interact with the array first before you can get to the points that make up the array. Next, you will work with a polyline feature class.

4 In PythonWin, create a new Python script and save as vertices.py **to the Results folder for exercise 8.**

5 Enter the following code:

```
import arcpy
from arcpy import env
env.workspace = "C:/EsriPress/Python/Data/Exercise08"
fc = "rivers.shp"
cursor = arcpy.da.SearchCursor(fc, (["OID@", "SHAPE@"]))
for row in cursor:
    print("Feature {0}: ".format(row[0]))
    for point in row[1].getPart(0):
        print("{0}, {1}".format(point.X, point.Y))
```

6 Save and run the script.

This prints a pair of x,y coordinates for each vertex in each feature, as follows:

```
Feature 0:
745054.499005 2324903.18355
745122.149446 2324835.9415
...
Feature 1:
747821.018772 2317111.87693
746909.057058 2317254.89224
...
Feature 2:
753597.862134 2315541.91647
753757.716791 2319122.62937
...
Feature 3:
751089.22614 2314030.50506
749458.383228 2315289.96843
...
```

This script works for polyline and polygon feature classes but does not work for multipart features, which you will work with next.

Work with multipart features

For multipart features, cursors return an array containing multiple arrays of point objects. Working with multipart features therefore requires an extra iteration over the parts of each feature.

The `isMultipart` property of the geometry object can be used to determine whether a particular feature is multipart.

1 In PythonWin, create a new Python script and save as multipart.py **to the Results folder for exercise 8.**

2 Enter the following code:

```
import arcpy
from arcpy import env
env.workspace = "C:/EsriPress/Python/Data/Exercise08"
fc = "dams.shp"
cursor = arcpy.da.SearchCursor(fc, ["OID@", "SHAPE@"])
for row in cursor:
    if row[1].isMultipart:
        print("Feature {0} is multipart.".format(row[0]))
    else:
        print("Feature {0} is single part.".format(row[0]))
```

3 Save and run the script.

This prints whether each feature in the features class is multipart or single part, as follows:

```
Feature 0 is single part.
Feature 1 is single part.
Feature 2 is single part.
Feature 3 is single part.
Feature 4 is single part.
...
```

The results are single part for each feature, because each feature consists of a single point—that is, a dam.

4 Modify the script as follows:

```
fc = "Hawaii.shp"
```

5 Save and run the script.

This confirms that the feature class representing Hawaii is a multipart polygon, as follows:

```
Feature 0 is multipart.
```

The `partCount` property can be used to determine the number of parts, which you'll do next.

6 Modify the first `print` statement following the `if` statement as follows:

```
    print("Feature {0} is multipart and has {1} parts.".format(row[0], ➔
  ➔ str(row[1].partCount)))
```

7 Save and run the script.

This prints the number of parts for every multipart feature, as follows:

```
Feature 0 is multipart and has 11 parts.
```

Because the geometry object for multipart features consists of an array containing multiple arrays of point objects, it is necessary to iterate over all parts of a feature to obtain its geometry. You need a loop to iterate over the arrays in the array, and then you need a loop to iterate over each point in the point array.

8 Modify the script as follows:

```
import arcpy
from arcpy import env
env.workspace = "C:/EsriPress/Python/Data/Exercise08"
fc = "Hawaii.shp"
cursor = arcpy.da.SearchCursor(fc, ["OID@", "SHAPE@"])
for row in cursor:
    print("Feature {0}: ".format(row[0]))
    partnum = 0
    for part in row[1]:
        print("Part {0}: ".format(partnum))
        for point in part:
            print("{0}, {1}".format(point.X, point.Y))
        partnum += 1
```

9 Save and run the script.

This prints the part number of each feature, followed by pairs of x,y coordinates of the vertices. In the case of the feature class Hawaii.shp, there is one feature with 11 parts, as follows:

```
Feature 0:
Part 0:
829161.23775 2245088.48637
829562.014007 2244825.55006
...
Part 1:
757434.846245 2276341.38313
757032.135863 2276024.28579
...
Part 2:
710001.620203 2315999.27091
712130.145201 2315404.37626
...
```

Write geometries

Insert and update cursors can be used to create new features and update existing features, respectively. Point objects can be created to set the geometry of these new or updated features.

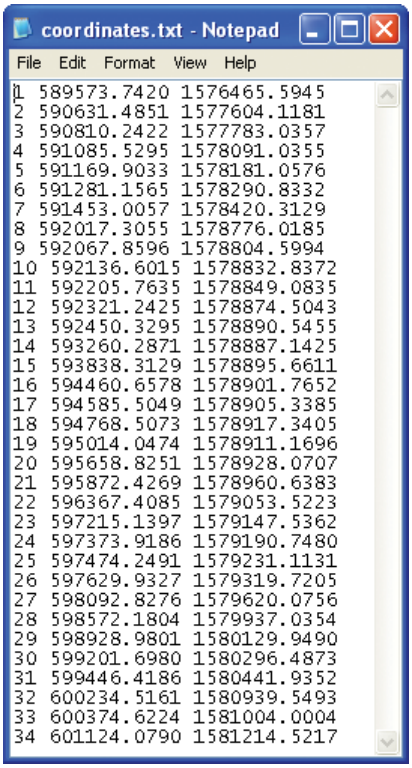
In the next example, you will use an insert cursor to create a new polyline feature from a list of coordinates.

1 Start Notepad. Typically, you can get there by clicking the Start button and then, on the Start menu, click All Programs > Accessories > Notepad.

2 On the Notepad menu bar, click File > Open and browse to the Exercise08 folder. Select the coordinates.txt file and click Open. ➔

The file consists of 34 pairs of x,y coordinates, with each pair preceded by an ID number and separated by a space. First, you will create a new empty polyline feature class, and then use this list of coordinates to create a new polyline.

3 Close Notepad.



4 **In PythonWin, create a new Python script and save as** create.py **to the Results folder for exercise 8.**

5 **Enter the following code.**

```
import arcpy
import fileinput
import string
import os
from arcpy import env
env.workspace = "C:/EsriPress/Python/Data/Exercise08"
env.overwriteOutput = True
outpath = "C:/EsriPress/Python/Data/Exercise08"
newfc = "Results/newpolyline.shp"
arcpy.CreateFeatureclass_management(outpath, newfc, "Polyline")
```

6 **Save and run the script.** This creates a new empty polyline feature class.

7 **Start ArcMap. Open the Catalog window. Navigate to the Results folder for exercise 8 and confirm that the file** newpolyline.shp **has been created.**

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

Next, you will read the text file.

9 **In the create.py script, add the following line of code:**

```
infile = "C:/EsriPress/Python/Data/Exercise08/coordinates.txt"
```

Note: *The full path is needed here because the workspace applies to only ArcPy functions and classes.*

The coordinates in this text file will be used to create the vertices for a polyline. This requires the use of an insert cursor to create new rows and an array object to contain the point objects.

10 **Add the following lines of code:**

```
cursor = arcpy.da.InsertCursor(newfc, ["SHAPE@"])
array = arcpy.Array()
```

Next, the script needs to read through the text file, parse the text into separate strings, and iterate over the text file to create a point object for every line in the text file.

11 Add the following lines of code:

```
for line in fileinput.input(infile):
    ID, X, Y = string.split(line," ")
    array.add(arcpy.Point(X, Y))
cursor.insertRow([arcpy.Polyline(array)])
fileinput.close()
del cursor
```

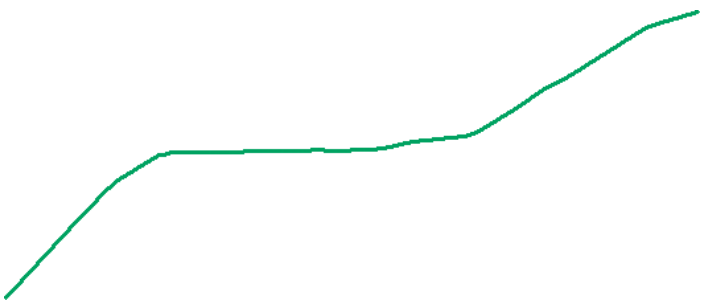
The completed script looks like the example in the figure.

newpoly.py

```
import arcpy
import fileinput
import string
import os
from arcpy import env
env.workspace = "C:/EsriPress/Python/Data/Exercise08"
env.overwriteOutput = True
outpath = "C:/EsriPress/Python/Data/Exercise08"
newfc = "Results/newpolyline.shp"
arcpy.CreateFeatureclass_management(outpath, newfc, "Polyline")
infile = "C:/EsriPress/Python/Data/Exercise08/coordinates.txt"
cursor = arcpy.da.InsertCursor(newfc, ["SHAPE@"])
array = arcpy.Array()
for line in fileinput.input(infile):
    ID, X, Y = string.split(line," ")
    array.add(arcpy.Point(X, Y))
cursor.insertRow([arcpy.Polyline(array)])
fileinput.close()
del cursor
```

12 Save and run the script.

13 Start ArcMap. Open the Catalog window and browse to the Results folder for exercise 8. Drag newpolyline.shp into the map document.



When adding the shapefile to the map document, you may get an error related to the spatial reference because this has not been set. This could be added to the code using the Create Feature Classes tool or as a separate line of code—for example, using the Define Projection tool.

Challenge exercises

Challenge 1

Write a script that creates a new polygon feature class containing a single (square) polygon with the following coordinates: (0, 0), (0, 1,000), (1,000, 0), and (1,000, 1,000).

Challenge 2

Write a script that determines the perimeter (in meters) and area (in square meters) of each of the individual islands of the Hawaii.shp feature class. Recall that this is a multipart feature class.

Challenge 3

Write a script that creates an envelope polygon feature class for the Hawaii.shp feature class. There is actually a tool that accomplishes this called Minimum Bounding Geometry. You can look at the tool to get some ideas, but your script needs to work directly with the geometry properties.

Exercise 9
Working with rasters

List the rasters

The `ListRasters` function can be used to list all the rasters in a workspace.

1 **Start PythonWin. Create a new Python script and save as** listrasters.py **to the C:\EsriPress\Python\Data\Exercise09\Results folder.**

2 **Enter the following code:**

```
import arcpy
from arcpy import env
env.workspace = "C:/EsriPress/Python/Data/Exercise09"
rasterlist = arcpy.ListRasters()
for raster in rasterlist:
    print raster
```

3 **Save and run the script.**

Running the script prints a list of rasters to the current workspace, as follows:

```
elevation
landcover.tif
tm.img
```

In this case, elevation is a raster in Esri GRID format and therefore has no file extension.

Describe the rasters

The `Describe` function can be used to describe raster properties.

1 **In PythonWin, create a new Python script and save as** describerasters.py **to the Results folder for exercise 9.**

2 **Enter the following code:**

```
import arcpy
from arcpy import env
env.workspace = "C:/EsriPress/Python/Data/Exercise09"
raster = "tm.img"
desc = arcpy.Describe(raster)
print "Raster base name is: " + desc.basename
print "Raster data type is: " + desc.dataType
print "Raster file extension is: " + desc.extension
```

3 **Save and run the script.**

Running the script prints a number of general raster properties, as follows:

```
Raster base name is: tm
Raster data type is: RasterDataset
Raster file extension is: img
```

More specific properties depend on whether the raster data element is a raster dataset, raster band, or raster catalog. The tm.img raster is a raster dataset, which makes it possible to access additional properties.

4 **Add the following print statements to the script:**

```
print "Raster spatial reference is: " + desc.spatialReference.name
print "Raster format is: " + desc.format
print "Raster compression type is: " + desc.compressionType
print "Raster number of bands is: " + str(desc.bandCount)
```

Notice that the band Count property which consists of a number and is combined using the plus sign (+), needs to be converted into a string for proper printing.

5 **Save and run the script.**

Running the script prints additional properties that are unique to raster datasets:

```
Raster base name is: tm
Raster data type is: RasterDataset
Raster file extension is: img
Raster spatial reference is: GCS_North_American_1983
Raster format is: IMAGINE Image
Raster compression type is: RLE
Raster number of bands is: 3
```

6 **Modify the script as follows:**

```
raster = "landcover.tif"
```

7 **Save and run the script.** In contrast to the tm.img raster, landcover.tif is a single-band raster. Additional raster properties can be accessed for individual raster bands. For single-band rasters, this is implicit, and the raster band does not need to be specified.

8 **Modify the script as follows:**

```
import arcpy
from arcpy import env
env.workspace = "C:/EsriPress/Python/Data/Exercise09"
raster = "landcover.tif"
desc = arcpy.Describe(raster)
x = desc.meanCellHeight
y = desc.meanCellWidth
spatialref = desc.spatialReference
units = spatialref.linearUnitName
print "The raster resolution is " + str(x) + " by " + str(y) + " " +
units + "."
```

9 **Save and run the script.**

Running the script prints the cell size in the units of the coordinate system of the raster, as follows:

```
The raster resolution is 30.0 by 30.0 Meter.
```

For multiband rasters, however, individual bands need to be specified.

10 Modify the script as follows:

```
import arcpy
from arcpy import env
env.workspace = "C:/EsriPress/Python/Data/Exercise09"
rasterband = "tm.img/Layer_1"
desc = arcpy.Describe(rasterband)
x = desc.meanCellHeight
y = desc.meanCellWidth
spatialref = desc.spatialReference
units = spatialref.angularUnitName
print "The raster resolution of Band 1 is " + str(x) + " by " + str(y) ➔
➔ + " " + units + "."
```

11 Save and run the script.

Running the script prints the raster resolution in the units of the coordinate system. Because the tm.img raster is in a geographic coordinate system, the units are angular units, not linear units, as follows:

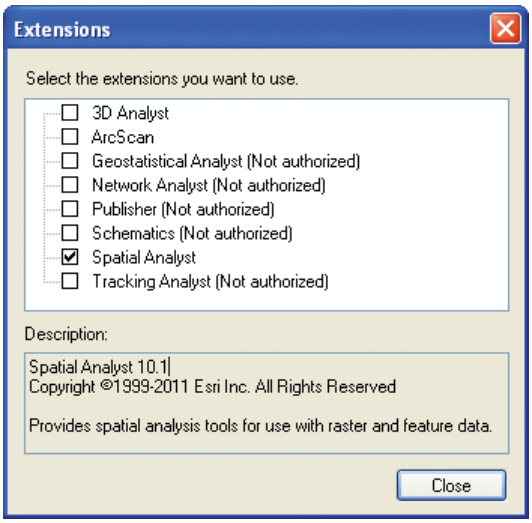
```
The raster resolution of Band 1 is 0.000277778 by 0.000277778 Degree.
```

Use raster objects in geoprocessing

To use the Spatial Analyst geoprocessing tools, you need an ArcGIS Spatial Analyst license. When working in the Python window, you can set this from within ArcMap.

1 Start ArcMap. On the menu bar, click Customize > Extensions.

2 On the Extensions dialog box, activate the ArcGIS Spatial Analyst extension by selecting that check box. ➔



3 Click Close. In working with raster datasets in Python scripting it is common to work with raster objects. Most raster geoprocessing tools return raster objects, which can be saved as rasters when necessary.

4 Open the Python window. This will allow you to see immediate results from running each line of code.

5 Run the following code:

```
>>> from arcpy import env
>>> env.workspace = "C:/EsriPress/Python/Data/Exercise09"
>>> outraster = arcpy.sa.Slope("elevation")
```

The new raster outraster is added to the ArcMap table of contents. Next, you will determine the permanent state of the raster.

6 Run the following code:

```
>>> desc = arcpy.Describe(outraster)
>>> print desc.permanent
```

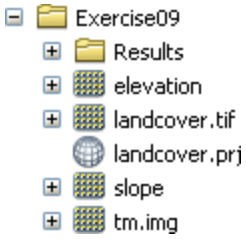
The result is `False`.

The new raster is only temporary. You can also determine the raster's status by examining the layer properties in ArcMap. When you exit ArcMap without saving the map document, the temporary raster will be deleted. The `save` method can be used to make the raster permanent.

7 Run the following code:

```
>>> outraster.save("slope")
```

8 Open the Catalog window in ArcMap. Navigate to the C:\EsriPress\Python\Data\Exercise09 folder and confirm that the slope raster has been saved.



In working with rasters, it is common to import all the functions of the `arcpy.sa` module, which you will do next. This module shortens the code to call geoprocessing tools.

9 Run the following code:

```
>>> from arcpy.sa import *
>>> outraster2 = Aspect("elevation")
>>> outraster2.save("aspect")
```

>>> TIP

To see the result in the Catalog window, right-click a folder (in this case, the Exercise09 folder) and click Refresh. This makes any newly added data files visible.

Use map algebra operators

The `arcpy.sa` module contains a number of map algebra operators, which you'll use next. These operators make it easier to write map algebra expressions in Python.

1 **Run the following code:**

```
>>> elevraster = arcpy.Raster("elevation")
```

Running this code creates a raster object by referencing a raster on disk. Using raster objects makes it easier to use raster datasets in code.

2 **Run the following code:**

```
>>> outraster3 = elevraster * 3.281
>>> outraster3.save("elev_ft")
```

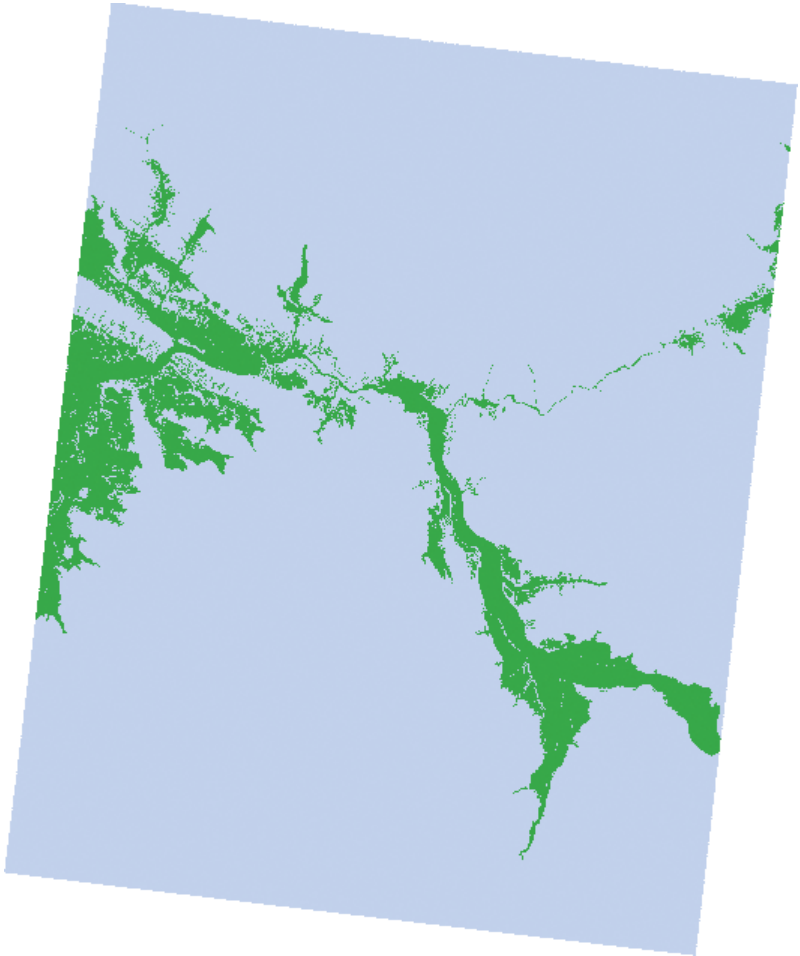
Running this code converts the elevation values from meters to feet and saves the raster object as a permanent raster. The map algebra operator (`*`) is used instead of the Times tool to make the code shorter. Map algebra operators include arithmetic, bitwise, Boolean, and relational operators.

3 **Run the following code:**

```
>>> slope = Slope(elevraster)
>>> goodslope = slope < 20
>>> goodelev = elevraster < 2000
>>> goodfinal = goodslope & goodelev
>>> goodfinal.save("final")
```

Running this code creates a new raster indicating slope less than 20 degrees and elevation less than 2,000 meters.

The result of the code, as shown in the figure, illustrates how map algebra operators can be used to carry out a series of geoprocessing operations. All the outputs are temporary raster objects, and only the final result is saved.



Work with classes to define raster tool parameters

Many raster tools have parameters that have a varying number of arguments. The `arcpy.sa` module has a number of classes to make it easier to work with these parameters.

You will be working with a stand-alone script, and the first task is to make sure a license for the Spatial Analyst extension is available.

1 Start PythonWin. Create a new Python script and save as slope.py **to the Results folder for exercise 9.**

2 Enter the following code:

```
import arcpy
from arcpy import env
from arcpy.sa import *
env.workspace = "C:/EsriPress/Python/Data/Exercise09"
if arcpy.CheckExtension("spatial") == "Available":
    arcpy.CheckOutExtension("spatial")
    outraster = arcpy.sa.Slope("elevation", "PERCENT_RISE")
    outraster.save("slope_per")
    arcpy.CheckInExtension("spatial")
else:
    print "Spatial Analyst license is not available."
```

3 Save and run the script.

In the Catalog window in ArcMap, confirm that the **slope_per** raster has been created. You now have a basic stand-alone script for working with tools from the `arcpy.sa` module. Next, you will use the Reclassify tool with the `RemapRange` class as one of the parameters.

4 Save your slope.py script as reclass.py**.**

5 In the reclass.py script, replace the two lines of code that pertained to creating and saving the slope raster with the following, making sure to maintain the indentation of these lines:

```
myremap = RemapRange([[1000,2000,1], [2000,3000,2], [3000,4000,3]])
outreclass = Reclassify("elevation", "VALUE", myremap)
outreclass.save("elev_recl")
```

6 Save and run the script.

7 In the Catalog window in ArcMap, confirm that the elev_recl **raster has been created.**

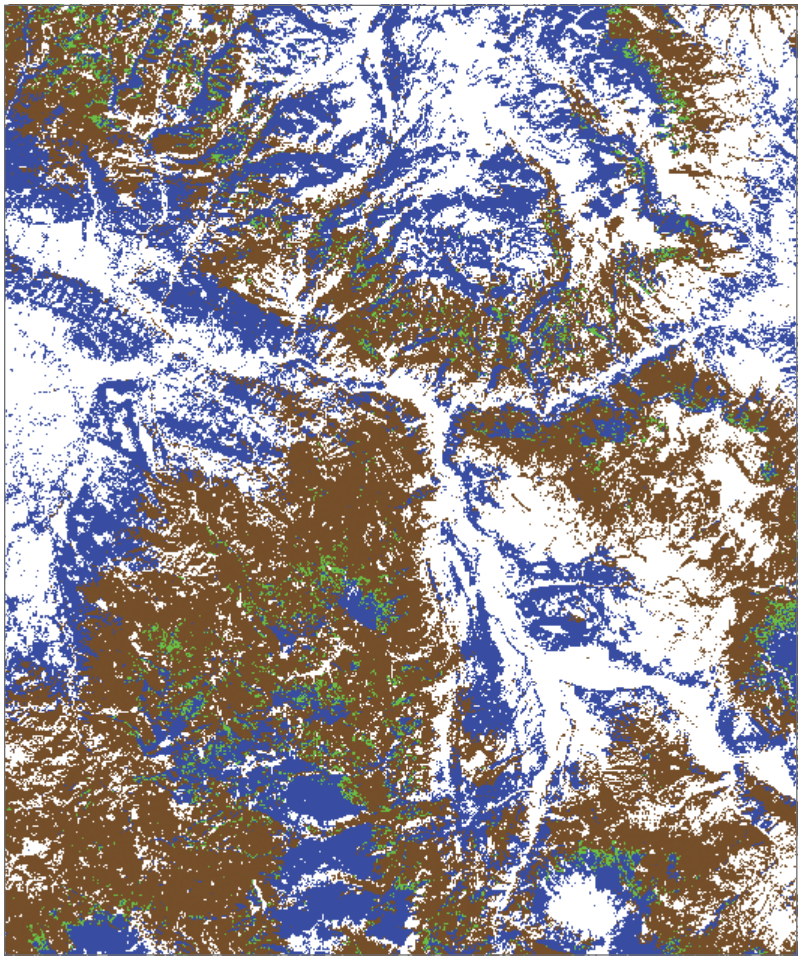
For discrete data, such as land cover, the `RemapValue` class is commonly used.

8 In the reclass.py script, replace the three lines of code that pertained to the reclassification of the elevation raster with the following:

```
myremap = RemapValue([[41,1], [42,2], [43,3]])
outreclass = Reclassify("landcover.tif", "VALUE", myremap, "NODATA")
outreclass.save("lc_recl")
```

9 Save and run the script.

10 In the Catalog window in ArcMap, confirm that the lc_recl **raster has been created.**



Other commonly used classes in raster-based analysis are the neighborhood classes, which you'll work with next.

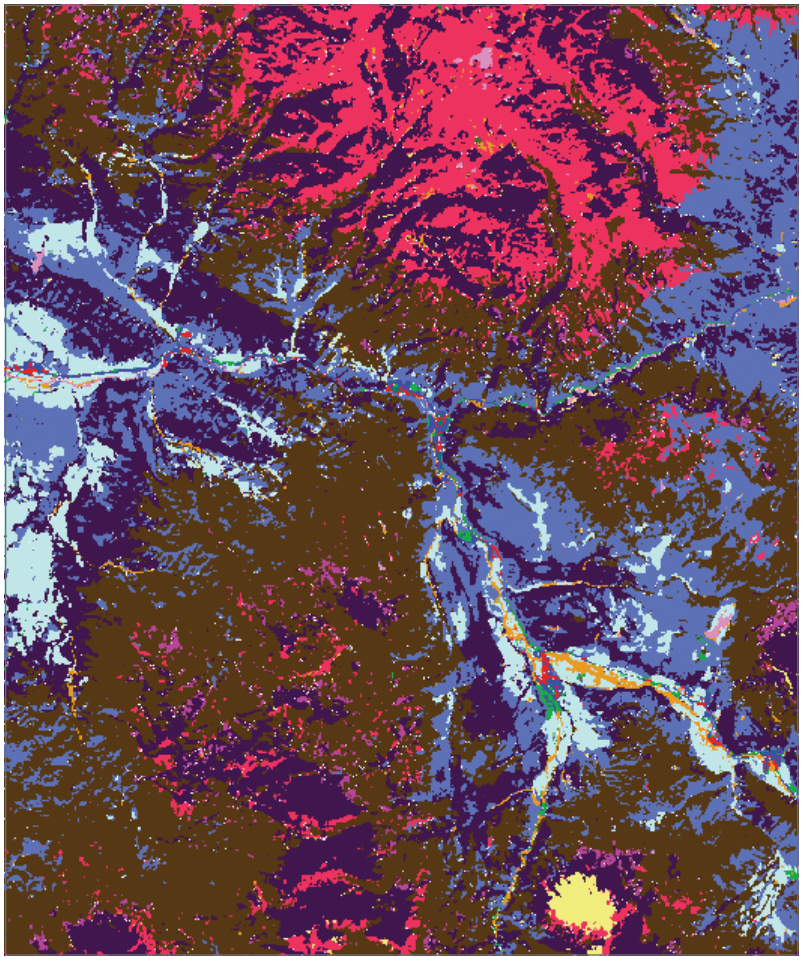
11 Save your reclass.py script as neighborhood.py**.**

12 In the neighborhood.py script, replace the three lines of code that pertained to the reclassification of the land cover raster with the following:

```
mynbr = NbrCircle(3, "CELL")
outraster = FocalStatistics("landcover.tif", mynbr, "MAJORITY")
outraster.save("lc_nbr")
```

13 Save and run the script.

14 In the Catalog window in ArcMap, confirm that the lc_nbr raster has been created.



Challenge exercises

Challenge 1

Create a script that determines what areas meet the following conditions:

- Moderate slope—between 5 and 20 degrees
- Southerly aspect—between 150 and 270 degrees
- Forested—land cover types of 41, 42, or 43

Be sure to use the map algebra expressions of the `arcpy.sa` module.

Challenge 2

Write a script that copies all the rasters in a workspace to a new file geodatabase. You can use the rasters in the Exercise09 folder as an example.

Part 3
Carrying out specialized tasks

Exercise 10
Map scripting

Open and save a map document

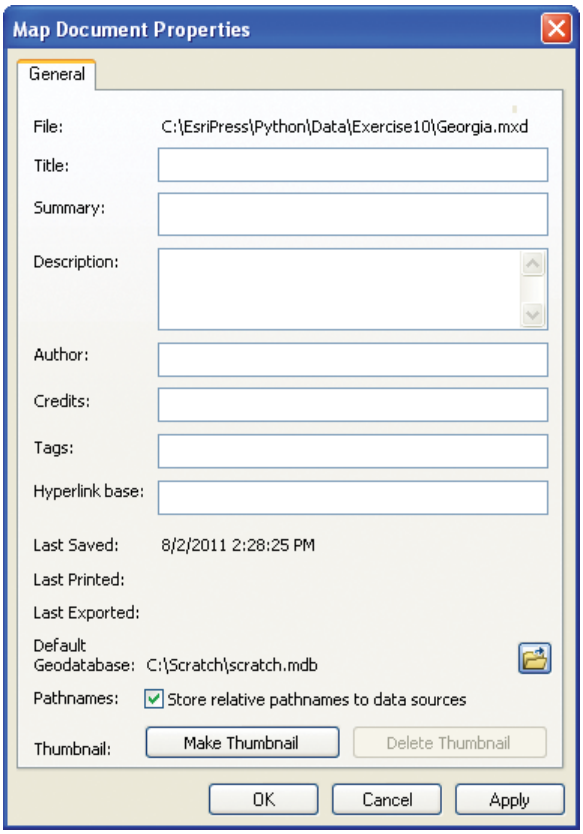
Map scripting can be used to open a map document and change any of the map document properties.

1 **Start ArcMap. Open the map document C:\EsriPress\Python\Data\Exercise10\Georgia.mxd.**

2 **On the ArcMap menu bar, click File > Map Document Properties.** Notice that most of the properties are blank, as shown in the figure. ➔

3 **Click OK to close the Map Document Properties dialog box.**

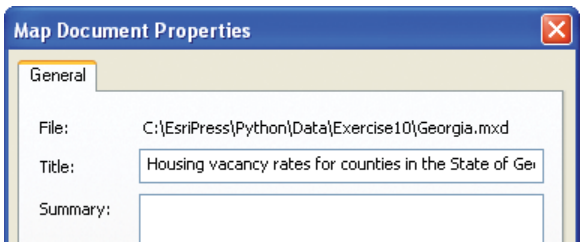
4 **On the Standard toolbar, click the Python button to open the Python window.**



5 Enter and run the following code:

```
>>> mapdoc = arcpy.mapping.MapDocument("CURRENT")
>>> mapdoc.title = "Housing vacancy rates for counties in the State of
Georgia, 2000"
>>> mapdoc.save()
>>> del mapdoc
```

6 On the ArcMap menu bar, click File > Map Document Properties.
Notice that the Title property has been modified, as shown in the figure.



7 Close ArcMap.

Work with data frames

Map scripting can be used to access and modify the properties of one or more data frames in a map document. In the first example, you will read the names of all the data frames.

1 Start PythonWin. Create a new Python script and save as dflist.py **to the C:\EsriPress\Python\Data\Exercise10\Results folder.**

2 Enter the following code:

```
import arcpy
mxd = "C:/EsriPress/Python/Data/Exercise10/Austin_TX.mxd"
mapdoc = arcpy.mapping.MapDocument(mxd)
listdf = arcpy.mapping.ListDataFrames(mapdoc)
for df in listdf:
    print df.name
del mapdoc
del listdf
```

3 Save and run the script.

Running the script prints the names of three data frames to the Interactive Window:

```
Facilities
Street Trees
Parks
```

You can also modify the data frame properties. In the next example, you will modify selected properties so they are identical for all data frames in the map document.

4 Start ArcMap. Open the map document C:\EsriPress\Python\Data\Exercise10\Austin_TX.mxd.

5 Briefly examine the layers and the properties of each data frame.

Notice that the extent is different for each, and the coordinate system for the Parks data frame is different from the other two. Next, you will use scripting to modify these properties and make them consistent.

6 Close ArcMap.

7 In PythonWin, create a new Python script and save as dfproperties.py **to the Results folder for exercise 10.**

8 Enter the following code:

```
import arcpy
mxd = "C:/EsriPress/Python/Data/Exercise10/Austin_TX.mxd"
mapdoc = arcpy.mapping.MapDocument(mxd)
dataset = "C:/EsriPress/Python/Data/Exercise10/Austin/base.shp"
spatialref = arcpy.Describe(dataset).spatialReference
extent = arcpy.Describe(dataset).extent
for df in arcpy.mapping.ListDataFrames(mapdoc):
    df.spatialReference = spatialref
    df.panToExtent(extent)
    df.scale = 15000
mapdoc.save()
del mapdoc
```

9 Save and run the script. Running this script sets the spatial reference, extent, and scale for all the data frames in the map document.

10 Start ArcMap. Open Austin_TX.mxd.

11 Confirm that the data frame properties have been modified.

Work with map layers

Map scripting can also be used to work with map layers. In the next example, you will create a list of all the layers in a map document and modify the properties of a specific layer.

1 In PythonWin, create a new Python script and save as maplayers.py **to the Results folder for exercise 10.**

2 Enter the following code:

```
import arcpy
mxd = "C:/EsriPress/Python/Data/Exercise10/Austin_TX.mxd"
mapdoc = arcpy.mapping.MapDocument(mxd)
for df in arcpy.mapping.ListDataFrames(mapdoc):
    print "Data frame " + df.name + " contains the following layers:"
    lyrlist = arcpy.mapping.ListLayers(mapdoc, "", df)
    for lyr in lyrlist:
        print lyr.name
del mapdoc
```

3 Save and run the script.

This prints the name of each data frame followed by the layers in each data frame:

```
Data frame Facilities contains the following layers:
addresses
facilities
sidewalks
base
Data frame Street Trees contains the following layers:
sidewalks
trees
buildings
base
Data frame Parks contains the following layers:
parks
base
```

Next, you will modify the properties of a specific layer in the map document.

4 **In ArcMap, make sure Austin_TX.mxd is still open. Activate the Parks data frame, notice how the labels of the parks layer are turned off, and turn off the parks layer.**

5 **Save Austin_TX.mxd.**

6 **Close ArcMap.**

7 **In PythonWin, create a new Python script and save as** lyrproperties.py **to the Results folder for exercise 10.**

8 **Enter the following code:**

```
import arcpy
mxd = "C:/EsriPress/Python/Data/Exercise10/Austin_TX.mxd"
mapdoc = arcpy.mapping.MapDocument(mxd)
lyrlist = arcpy.mapping.ListLayers(mapdoc)
for lyr in lyrlist:
    if lyr.name == "parks":
        print lyr.name
        lyr.visible = True
        lyr.showLabels = True
mapdoc.save()
del mapdoc
del lyrlist
```

9 **Save and run the script.**

10 **Start ArcMap. Open Austin_TX.mxd.**

11 **Confirm that the properties of the parks layer have been modified.**

12 **Close ArcMap.**

Work with page layout elements

Map scripting also makes it possible to work with page layout elements. In the next examples, you will create a list of all the elements of a page layout and modify one of the elements.

1 **In PythonWin, create a new Python script and save as** elemlist.py **to the Results folder for exercise 10.**

2 Enter the following code:

```
import arcpy
mxd = "C:/EsriPress/Python/Data/Exercise10/Georgia.mxd"
mapdoc = arcpy.mapping.MapDocument(mxd)
elemlist = arcpy.mapping.ListLayoutElements(mapdoc)
for elem in elemlist:
    print elem.name + " " + elem.type
del mapdoc
```

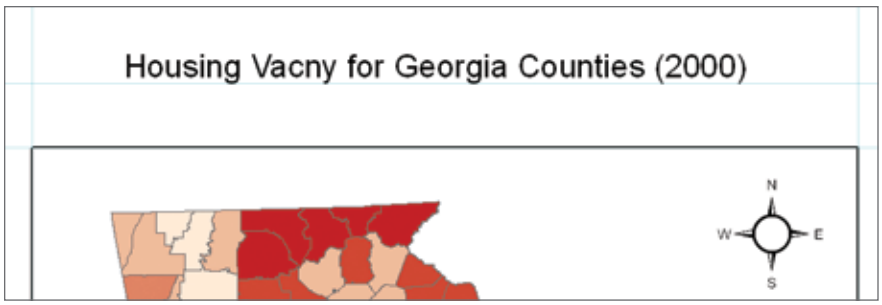
3 Save and run the script.

This prints the name and type of each page layout element, as follows:

```
Title TEXT_ELEMENT
Stepped Scale Line MAPSURROUND_ELEMENT
North Arrow MAPSURROUND_ELEMENT
Legend LEGEND_ELEMENT
Vacancy DATAFRAME_ELEMENT
```

Next, you will modify one of the elements.

4 Start ArcMap. Open Georgia.mxd. Notice that there is a typo in the title where the word "vacancy" is misspelled, as shown in the example in the figure.



5 Close ArcMap.

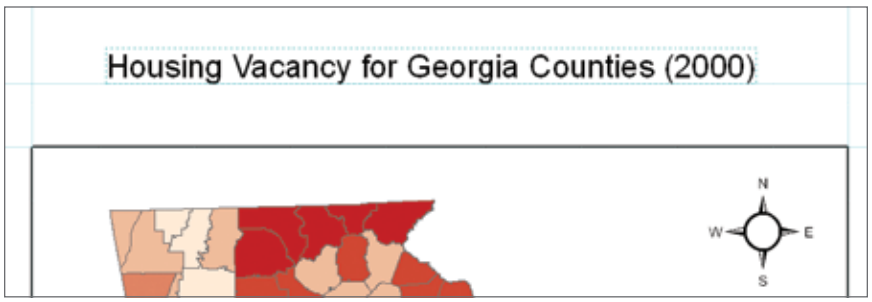
6 In PythonWin, create a new Python script and save as elemproperties.py **to the Results folder for exercise 10.**

7 Enter the following code:

```
import arcpy
mxd = "C:/EsriPress/Python/Data/Exercise10/Georgia.mxd"
mapdoc = arcpy.mapping.MapDocument(mxd)
elemlist = arcpy.mapping.ListLayoutElements(mapdoc)
title = elemlist[0]
title.text = "Housing Vacancy for Georgia Counties (2000)"
mapdoc.save()
del mapdoc
```

8 Save and run the script.

9 Start ArcMap. Open Georgia.mxd. Notice that the typo in the title has been corrected, as shown in the example in the figure.



Challenge exercise

Challenge 1

In ArcGIS Desktop Help, research the `AddLayer` function of the ArcPy mapping module and use it to write a script that adds the parks layer from the Parks data frame in Austin_TX.mxd to the other two data frames in the same map document.

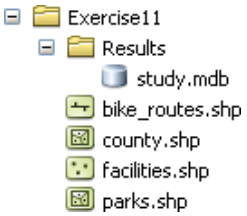
Exercise 11

Debugging and error handling

Examine syntax errors and exceptions

To start the exercise, first take a quick look at the data.

1 **Start ArcMap. Open the Catalog window. Navigate to the C:\EsriPress\Python\Data\Exercise11 folder.** ➔



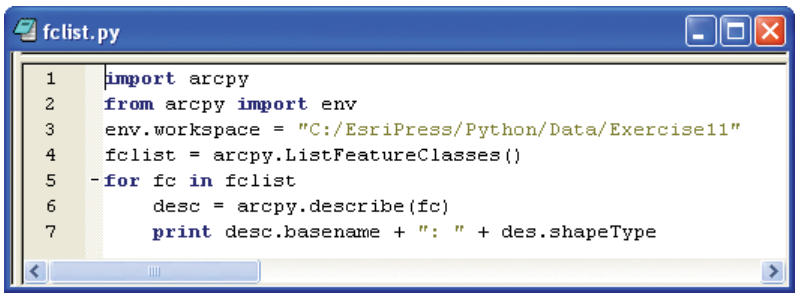
Notice that there are four shapefiles in this folder, plus one empty personal geodatabase in the Results folder.

There are also two scripts in the Exercise11 folder, which are not visible in ArcCatalog by default. These scripts contain errors, which you will fix in this exercise.

2 **Close ArcMap. There is no need to save the map document.**

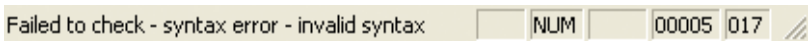
Syntax errors prevent code from being executed. In the following examples, you will identify some common syntax errors.

3 **Start PythonWin. On the menu bar, click File > Open, navigate to the Exercise11 folder, and open the script fclist.py.**



This script has a number of syntax errors. You may be able to identify them directly, but even so, it is useful to see how they can be found.

4 **With the cursor placed inside the script, click the Check button.** This displays an error message on the PythonWin status bar. The cursor is placed at the end of the line where the syntax error occurred: line 5, character position 17.



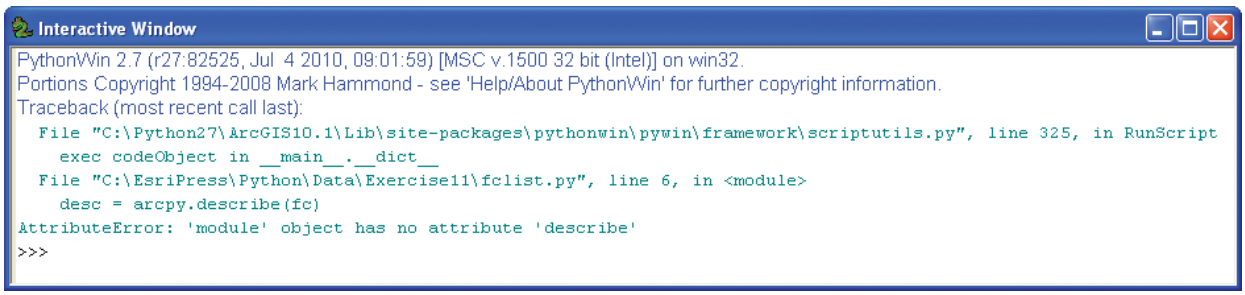
The error message does not report what the error is, but only where it occurs—you still have to identify the exact nature of the syntax error. In this case, the error is a missing colon (:) at the end of the line of code.

5 **Correct the code on line 5 of the script as follows:**

```
for fc in fclist:
```

6 **Save and run the script.** The script runs but is interrupted and an error message appears on the PythonWin status bar, as shown in the figure.

An error message is also printed to the Interactive Window, as shown in the figure.



An `AttributeError` exception was raised at line 6 in the script. The module `arcpy` does not have an attribute called `describe`. The correct spelling of the function is `Describe`. Remember that Python is case sensitive, for the most part.

7 **Correct the code on line 6 of the script as follows:**

```
desc = arcpy.Describe(fc)
```

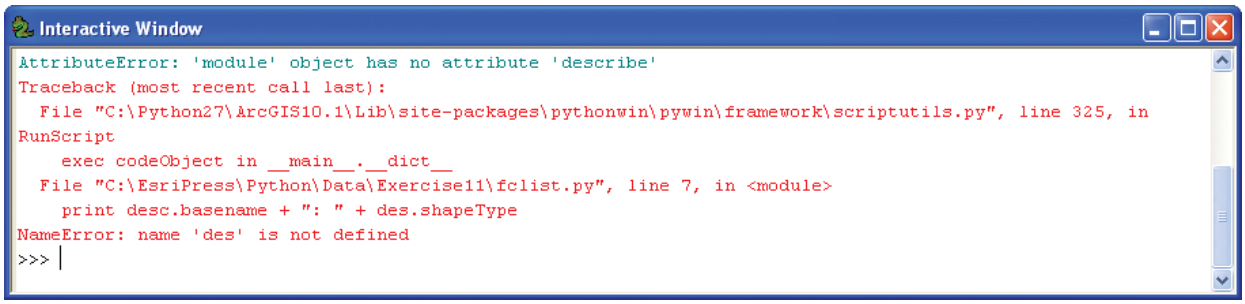
>>> TIP

Line numbering can be made visible in PythonWin by clicking View > Options from the menu bar. On the PythonWin Options dialog box, click the Editor tab and under Margin Widths, increase the size for Line Numbers—for example, to 30.

PythonWin also shows the position of the cursor within a script window by a designation on the far right of the status bar. The first number is the line number, and the second number is the character position.

8 Save and run the script. Again, an error message appears on the PythonWin status bar, and the details are printed to the Interactive Window.

Note: *For longer error messages, it is best to start reading them from the bottom up. For example, the last line shows the information about the exact error. Above that, it shows the line that's in error. And above that, it reports the line number as line 7.*

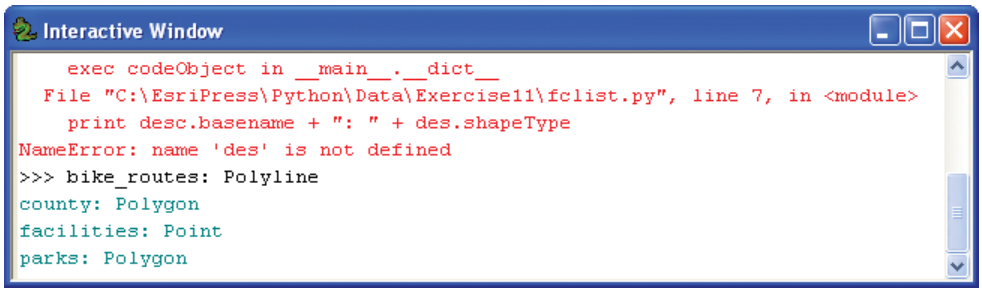


A `NameError` exception was raised on line 7 in the script. The name `des` is not defined. The correct spelling of the variable is `desc`.

9 Correct the code on line 7 of the script as follows:

```
print desc.basename + ": " + desc.shapeType
```

10 Save and run the script. This time the script runs correctly, and the result is printed to the Interactive Window.



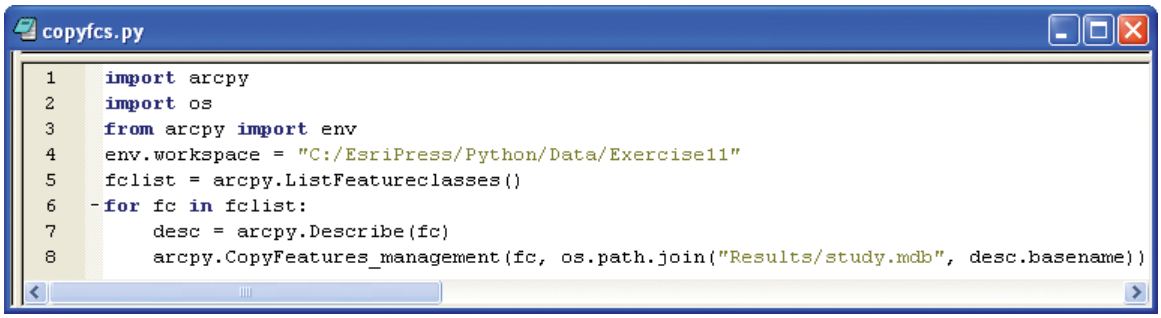
This example illustrates some of the most common errors in Python scripts: punctuation, capitalization, and spelling.

11 Close the fclist.py script.

Implement debugging procedures

Many errors can be identified simply by trying to run a script and investigating the error messages. An alternative is to step through your code line by line using the Python debugger.

1 On the PythonWin Standard toolbar, click the Open button, navigate to the Exercise11 folder, and open the script copyfcs.py.

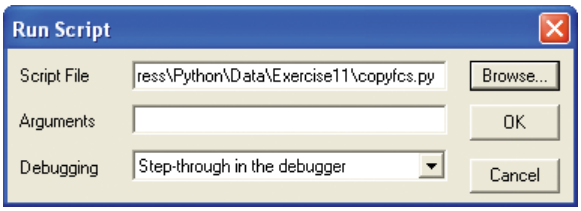


Take a moment to examine the code. The script creates a list of all the feature classes in a workspace and copies them to a personal geodatabase.

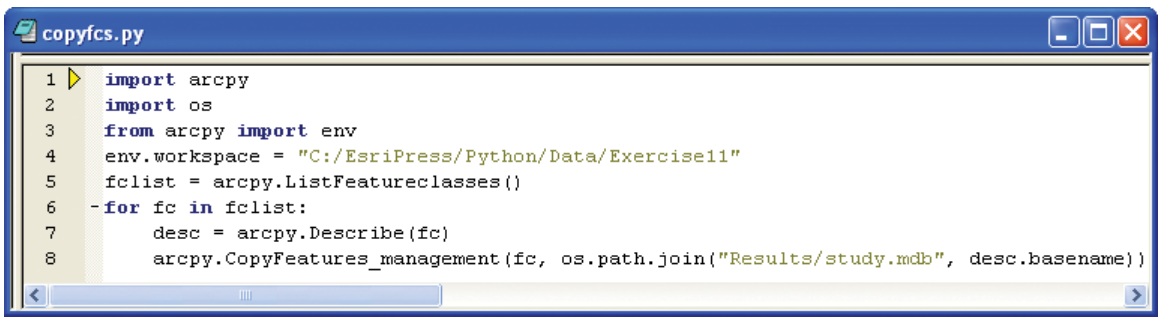
2 With your pointer placed inside the script, click the Check button. There are no syntax errors.

3 Click the Run button.

4 On the Run Script dialog box, under the Debugging options, click "Step-through in the debugger".



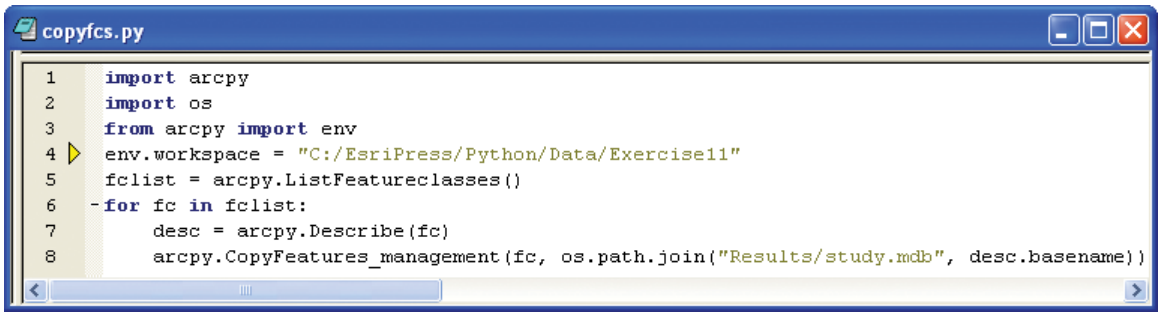
5 Click OK. This brings up the Debugger toolbar, and in the script, a yellow arrow points to the first line of executable code.



Using the Debugger tools, you can step through the code line by line. If any errors occur while running the code line by line, the error messages will be printed to the Interactive Window.

6 **On the Debugger toolbar, click the Step button.** This runs the first line of code, and the yellow arrow points to the second line of code.

7 **Click the Step button again twice.** The yellow arrow now points to the fourth line of code.



8 **Click the Step button again.** This opens the _base.py module, with the yellow arrow pointing to line 512 of the script, although the line number may change between different versions of Python. This is one of the built-in ArcPy scripts. When the workspace is set as part of the environment settings, this module is called. You probably don't want to step through this module but let the code run and return to your script, so that you can then continue to step through your own script.

9 **On the Debugger toolbar, click the Step Out button twice.**

10 **Close the _base.py script.** This brings you back to your own script, with the yellow arrow pointing to line 5.

To avoid opening other modules, you will use the Step Over button next, instead of Step. The Step Over button runs the current line of code, and if it includes any Python modules or functions, they will be run as well, and the cursor will return to the script itself.

11 **Click the Step Over button.**

This prints an error message to the Interactive Window, which follows in part:

```
AttributeError: 'module' object has no attribute 'ListFeatureclasses'
```

12 **On the Debugger toolbar, click the Close button.**

13 **Correct line 5 of the script as follows:**

```
fclist = arcpy.ListFeatureClasses()
```

14 **Save the script.**

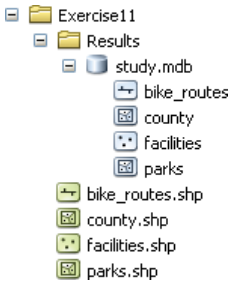
15 **Click the Run button.**

16 **On the Run Script dialog box, under the Debugging options, click "No debugging".** This runs the script, and no more errors are encountered. Successful execution is reported on the PythonWin status bar.

```
Script 'C:\EsriPress\Python\Data\Exercise11\copyfcs.py' returned exit code 0
```

Note: You need to stop the debugger so you can make changes to the script and run it again.

17 **Start ArcMap. Open the Catalog window. Navigate to the Exercise11 folder. Confirm that the geodatabase study.mdb contains four feature classes.** →



18 **Close ArcMap.**

Handle some exceptions

Many different types of errors can occur when running a script. Rather than just letting a script cause a runtime error, you can gain more control using certain error-handling procedures. The most widely used error-handling technique uses a `try-except` statement.

Consider the script from the preceding set of steps. What if the geodatabase study.mdb did not exist?

1 **In PythonWin, make sure the script copyfcs.py is still open.**

2 **On line 8, replace** `study.mdb` **with** `mydata.mdb`**.**

3 **Save and run the script.** This produces an error message on the PythonWin status bar, as shown in the figure.

```
Exception raised while running script  copyfcs.py
```

A more detailed error message is also printed to the Interactive Window, as follows:

```
ExecuteError: ERROR 000210: Cannot create output
Results/mydata.mdb\bike_routes
Failed to execute (CopyFeatures).
```

Next, you will trap this error using a `try-except` statement.

4 Modify the code, as follows:

```
import arcpy, os
from arcpy import env
try:
    env.workspace = "C:/EsriPress/Python/Data/Exercise11"
    fclist = arcpy.ListFeatureClasses()
    for fc in fclist:
        desc = arcpy.Describe(fc)
        arcpy.CopyFeatures_management(fc, os.path.join("Results/
➔ mydata.mdb", desc.basename))
except arcpy.ExecuteError:
    print arcpy.GetMessages(2)
except:
    print "There has been a nontool error."
```

5 Save and run the script. This time the script runs successfully.

```
Script 'C:\EsriPress\Python\Data\Exercise11\copyfcs.py' returned exit code 0
```

However, an error message is printed to the Interactive Window, as follows:

```
ExecuteError: ERROR 000210: Cannot create output
Results/mydata.mdb\bike_routes
Failed to execute (CopyFeatures).
```

Although the error message is the same as before, there is a very important difference: despite the error, the script ran successfully rather than resulting in a runtime error. This is a very important distinction. Say, for instance, that you called this script as a tool in ArcMap. If the script results in a runtime error, you may not find out what happened because the printed error message does not appear anywhere. With the use of the `try-except` statement, the script runs successfully, and the error message can be reported back to the tool that called the script.

Challenge exercises

Challenge 1

The following script contains a number of errors. Try to identify all four.

```
import arcpy
from arcpy import env
env.workspace = "C:/EsriPress/Python/Data/Exercise07"
FC = "airports.shp"
rows = arcpy.SearchCursor(fc)
fields = arcpy.ListFields(fc)
for field in fields:
    if fields.name == "NAME"
        for row in rows:
        print "Name = {0}".format(row.getValue(field.name))
```

Challenge 2

The following script contains a number of errors. Try to identify all six.

```
import arcpy
from arcpy import env
env.workspace = "C:/EsriPress/Python/Data\Exercise09"
raster = "landcover.tiff"
desc = arcpy.describe(raster)
x = desc.MeanCellHeight
y = desc.MeanCellWidth
spatialref = desc.spatialReference
units = spatialref.linearUnitName
print "Cells are" + str(x) + " by " + str(y) + " " + units + "."
```

Exercise 12

Creating Python functions and classes

Create Python functions

In this exercise, you will create a custom function that can be called from within the same script or from another script.

1 Start PythonWin. Create a new Python script and save as list.py to the C:\EsriPress\Python\Data\Exercise12\Results folder.

2 Enter the following code:

```
import arcpy
arcpy.env.workspace = "C:/EsriPress/Python/Data/Exercise12"
fields = arcpy.ListFields("streets.shp")
print fields
```

Running this code creates a list of fields.

3 Save and run the script.

Running the script prints the Python reference information for each field object in the streets shapefile. The code does not print any field names. The output prints as follows:

```
[<Field object at 0xfb9870[0xe021e90]>, <Field object at
0xe0d33f0[0xe021f38]>, <Field object at 0xe0d3570[0xe021f68]>,
<Field object at 0xe0d37b0[0xe104038]>, <Field object at
0xe0d3b90[0xe1040b0]>,...
```

To print the names of the field objects, you can use their `name` property. A `for` loop can be used to iterate over the list of fields.

4 Modify the code as follows:

```
import arcpy
arcpy.env.workspace = "C:/EsriPress/Python/Data/Exercise12"
fields = arcpy.ListFields("streets.shp")
namelist = []
for field in fields:
    namelist.append(field.name)
print namelist
```

5 Save and run the script.

Running the script prints the field names, as follows:

```
[u'FID', u'Shape', u'OBJECTID', u'PRE_TYPE', u'ST_NAME', u'STREET_
NAM', u'STREET_TYP', u'SUF_DIR', u'FULLNAME', ...]
```

Once you have the script to create a list of field names, you may want to use it again. You can do this by creating a custom function, which you'll do next.

6 Modify the code as follows:

```
import arcpy

def listfieldnames(table):
    fields = arcpy.ListFields(table)
    namelist = []
    for field in fields:
        namelist.append(field.name)
    return namelist
```

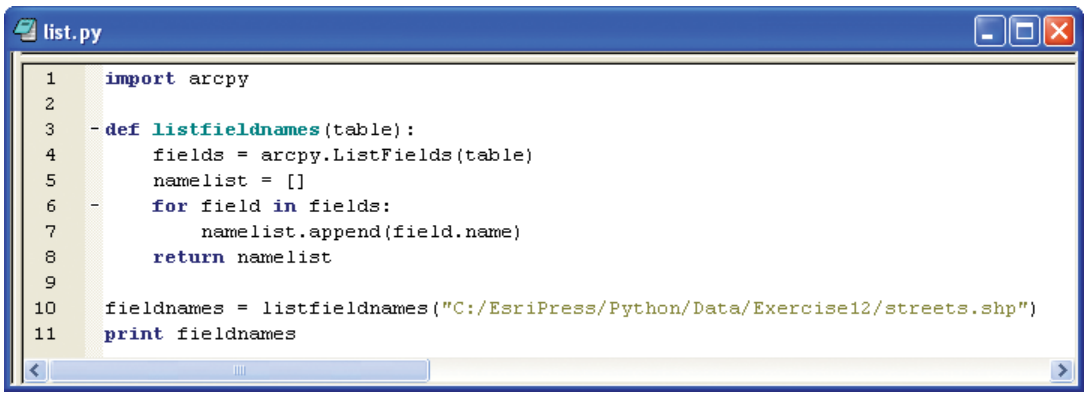
It is common to skip a line in a script before a custom function. This has no effect on running the script.

The block of code that creates the list of names is now defined as a function called `listfieldnames`. This function can now be called, for example, from within the same script. You'll do this next.

7 Add the following lines of code:

```
fieldnames = listfieldnames("C:/EsriPress/Python/Data/Exercise12/
streets.shp")
print fieldnames
```

The complete script now looks like the example in the figure.



8 Save and run the script. Running the script prints the list of field names to the Interactive Window.

Initially, it does not appear to do anything different from the earlier version of the script that did not define a custom function. However, once the function is created, it can also be called from another script. You will do that next.

Call functions from other scripts

1 In PythonWin, create a new Python script and save as myscript.py **to the Results folder for exercise 12.**

2 Enter the following code:

```
import arcpy
import list
arcpy.env.workspace = "C:/EsriPress/Python/Data/Exercise12"
fields = list.listfieldnames("streets.shp")
print fields
```

Running this script imports the `list` module, calls the `listfieldnames` function, and passes the name of a table as an argument to this function.

3 Save and run the script. The result is that the list of field names is printed twice to the Interactive Window.

What happened? When the myscript.py script called the `listfieldnames` function, it ran the list.py script. This script creates and then prints the list of field names. The function also returns the list of field names, and it is printed by the myscript.py script.

To avoid double printing, some additional code is needed, which you'll add next.

***Note:** If PythonWin had been left open with the list.py script open from the previous steps, the script would not have been loaded again, and the result would have printed only once.*

4 Open the list.py script and add the following line of code just before the line that starts with " `fieldnames ...`**":**

```
if __name__ == "__main__":
```

***Note:** There are two underscores in each spot.*

5 **Indent the last two lines of code.** The script should now look like the example in the figure.

list.py

```
1   import arcpy
2
3   def listfieldnames(table):
4       fields = arcpy.ListFields(table)
5       namelist = []
6       for field in fields:
7           namelist.append(field.name)
8       return namelist
9
10  if __name__ == "__main__":
11      fieldnames = listfieldnames("C:/EsriPress/Python/Data//Exercise12/streets.shp")
12      print fieldnames
```

6 **Save the list.py script.**

7 **Close PythonWin.**

8 **Start PythonWin and open the myscript.py script.**

9 **Without making any changes, run the script.** Running the script prints the list of field names only once to the Interactive Window.

Interactive Window

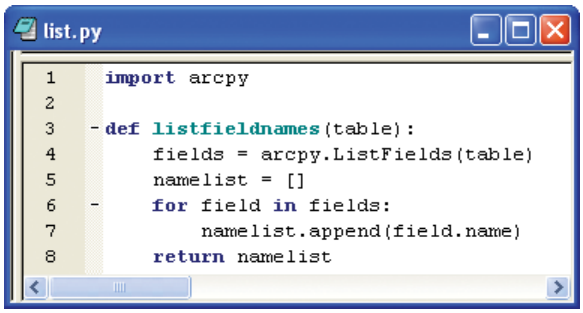
```
PythonWin 2.7 (r27:82525, Jul  4 2010, 09:01:59) [MSC v.1500 32 bit (Intel)] on win32.
Portions Copyright 1994-2008 Mark Hammond - see 'Help/About PythonWin' for further copyright information.
>>> [u'FID', u'Shape', u'OBJECTID', u'PRE_TYPE', u'ST_NAME', u'STREET_NAM', u'STREET_TYP', u'SUF_DIR',
u'FULLNAME', u'LOW', u'HIGH', u'L_ADD_FROM', u'L_ADD_TO', u'R_ADD_FROM', u'R_ADD_TO', u'L_PLACE', u
'R_PLACE', u'L_CITY', u'R_CITY', u'STATE', u'ZIP_RIGHT', u'ZIP_LEFT', u'ESN_LEFT', u'ESN_RIGHT', u
'L_PRECINCT', u'R_PRECINCT', u'L_SCHOOL', u'R_SCHOOL', u'REGION', u'EDIT_DATE', u'COMMENTS', u'STATUS',
 u'SURFACE_TY', u'SPEED', u'NUM_LANES', u'ROAD_ROW_W', u'ALIAS', u'RD_NUMBER', u'FUNCTIONAL', u
'LABEL_CLAS', u'ROADCLASS', u'CLASS', u'LINK_CLASS', u'LINK_NUM', u'SNOW_ROUTE', u'GARBAGE_RO', u
'RECYCLING_', u'STATE_AID_', u'STATE_AID1', u'MAINT_FUND', u'MAINT_OWNE', u'MAINT_BY', u'MAINT_DIST', u
'MAINT_DI_1', u'MAINT_TYPE', u'MAINT_WINT', u'MAINT_RURA', u'MAINT_LINK', u'HWY_CONTRO', u'HWY_SEGMEN',
 u'LOAD_RESTR', u'POSTAL_ROU', u'NAVSTREETS', u'CO_CODE', u'CO_NAME', u'PSAP_ID', u'ONEWAY', u
'TF_MINUTES', u'FT_MINUTES', u'F_ZLEV', u'T_ZLEV', u'FEET', u'Shape_len']
```

When the `myscript.py` script runs and calls the `listfieldnames` function, the `if __name__ == "__main__":` statement in the list.py script ensures that the next block of code is run only if the list.py script is run by itself.

The block of code following the `if __name__ == "__main__":` statement can be considered a "test." When the list.py script is run by itself, this test code allows you to check whether the custom function works correctly. However, it may not be necessary to keep this code

if the script is being used to store a custom function that will only be called from other scripts.

10 Modify the list.py script by removing the last three lines of code. The script should now look like the example in the figure.



11 Save and close the list.py and myscript.py scripts.

Work with classes

Classes allow you to group functions and variables together. Once created, classes make it possible to create objects that have specific properties as defined by these functions and variables.

Next, you will first consider a script to calculate property taxes, and then you will create a class to make this calculation more versatile.

1 In PythonWin, create a new Python script and save as assessment.py **to the Results folder for exercise 12.**

2 Enter the following code:

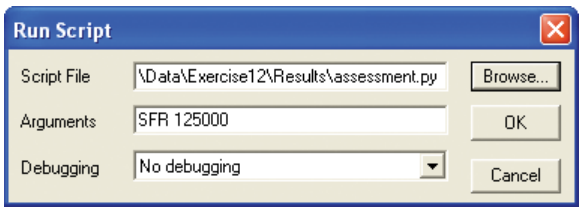
```
import sys
landuse = sys.argv[1]
value = int(sys.argv[2])

if landuse == "SFR":
    rate = 0.05
elif landuse == "MFR":
    rate = 0.04
else:
    rate = 0.02
assessment = value * rate
print assessment
```

This script calculates the property tax assessment based on variables for land use and the property value.

3 **Save the script.** The script requires two arguments.

4 **Click the Run button. On the Run Script dialog box, enter the arguments as shown in the figure.**



5 **Click OK.** Running the script prints the result of `6250.0` to the Interactive Window.

To automate this calculation for many different entries, you would read the values from a file and iterate over these values. Then you would have a few options to carry out the tax calculation. First, you can place the code within the iteration—that is, within the `for` loop or the `while` loop. Second, you can create a custom function in a separate script that does the calculation; when the function is called, the necessary arguments are passed, and the function returns a value. Third, you can create a class that contains the calculation as a method.

Next, you will take a look at the use of a custom function.

6 **In PythonWin, create a new Python script and save as** tax.py **to the Results folder for exercise 12.**

7 **Enter the following code:**

```
def taxcalc(landuse, value):
    if landuse == "SFR":
        rate = 0.05
    elif landuse == "MFR":
        rate = 0.04
    else:
        rate = 0.02
    assessment = value * rate
    return assessment
```

8 **Save your script.**

9 Create a new Python script and save as parcelCalc.py **to the Results folder for exercise 12.**

10 Enter the following code:

```
import tax
mytax = tax.taxcalc("SFR", 125000)
print mytax
```

11 Save and run the script. Running the script prints the result of 6250.0 to the Interactive Window.

Next, you will look at the use of a class.

12 Create a new Python script and save as parcelclass.py **to the Results folder for exercise 12.**

13 Enter the following code:

```
class Parcel:
    def __init__(self, landuse, value):
        self.landuse = landuse
        self.value = value

    def assessment(self):
        if self.landuse == "SFR":
            rate = 0.05
        elif self.landuse == "MFR":
            rate = 0.04
        else:
            rate = 0.02
        assessment = self.value * rate
        return assessment
```

14 Save your script.

15 Create a new Python script and save as parcelTax.py **to the Results folder for exercise 12.**

16 Enter the following code:

```
import parcelclass
myparcel = parcelclass.Parcel("SFR", 125000)
print "Land use: ", myparcel.landuse
mytax = myparcel.assessment()
print "Tax assessment: ", mytax
```

17 Save and run the script.

Running the script prints the following tax information to the Interactive Window:

```
Land use: SFR
Tax assessment: 6250.0
```

Although the use of the class accomplishes the same calculation as the custom function, a class is more versatile because it allows you to combine properties and functions. It would be relatively easy, for example, to expand the class with additional methods for different calculations, which could all be part of the same class.

Challenge exercises

Challenge 1

Create a custom function called **countstringfields** that determines the number of fields of type `string` in an input feature class. Create this function in a separate script (for example, **mycount.py**) that you call from another script (for example, **callingscript.py**). You can use the streets.shp feature class in the Exercise12 folder.

Challenge 2

You are given a feature class called parcels.shp located in the Exercise12 folder that contains the following fields: FID, Shape, Landuse, and Value. Modify the parceltax.py script so that it determines the property tax for each parcel and stores these values in a list. You should use the class created in the parcelclass.py script—the class can remain unchanged. Print the values of the final list as follows:

```
FID: <property tax>
```

Exercise 13
Creating custom tools

Examine the script

In this exercise, you will create a script tool that creates a random selection of features from an existing feature class and saves the result as a new feature class. First, examine the script.

1 **Start PythonWin. On the menu bar, click the Open button and browse to the script random_sample.py in your C:\EsriPress\Python\Data\Exercise13 folder.**

2 **Click OK.**

3 Examine the script.

random_sample.py

```
1  import arcpy
2  import random
3  from arcpy import env
4  env.overwriteoutput = True
5  inputfc = "C:/EsriPress/Python/Data/Exercise13/points.shp"
6  outputfc = "C:/EsriPress/Python/Data/Exercise13/Results/random.shp"
7  outcount = 50
8  desc = arcpy.Describe(inputfc)
9  inlist = []
10 randomlist = []
11 fldname = desc.OIDFieldName
12 rows = arcpy.SearchCursor(inputfc)
13 row = rows.next()
14 while row:
15     id = row.getValue(fldname)
16     inlist.append(id)
17     row = rows.next()
18 while len(randomlist) < outcount:
19     selitem = random.choice(inlist)
20     randomlist.append(selitem)
21     inlist.remove(selitem)
22 length = len(str(randomlist))
23 sqlexp = '"' + fldname + '"' + " in " + "(" + str(randomlist)[1:length - 1] + ")"
24 arcpy.MakeFeatureLayer_management(inputfc, "selection", sqlexp)
25 arcpy.CopyFeatures_management("selection", outputfc)
```

Notice that the names of the input and output feature classes and the count of random features to be selected are hard-coded into the script.

A few points to note about the script:

- The script creates a new empty list, `inlist`, and this is populated with the OID (object ID) values of all the features in the input feature class.
- The script creates a new empty list, `randomlist`, and this is populated with 50 OID values randomly selected from `inlist`.
- The output feature class is created by using the OID values in an SQL expression to select features from the input feature class. The SQL expression looks a bit complicated but essentially uses the `in` operator to query each feature in the input feature class as to whether its OID value is part of `randomlist`.

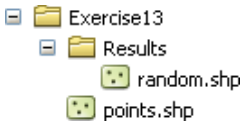
First, you will run the script to see what the output looks like.

4 Run the script.

5 Close PythonWin.

Next, you will examine the results.

6 Start ArcMap. Open the Catalog window. Navigate to the C:\EsriPress\Python\Data\Exercise13\Results folder. ➔



7 Drag the shapefiles points.shp (from the Exercise13 folder) and random.shp (from the Results folder for exercise 13) into the data frame. When you examine the contents of the two shapefiles, you will notice that the points.shp file contains 1,314 features, and the random.shp file contains only 50 features created through a random selection. If you were to run the script again, you would get a different random selection.

Next, instead of running the script from PythonWin, you will create a script tool so that it can be run from within an ArcGIS for Desktop application.

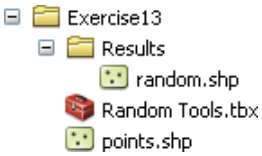
Create a custom tool

Script tools are located inside a toolbox. First, you will create a new custom toolbox. Then within that toolbox, you will create a new tool that makes use of the preceding script.

1 **In ArcMap, make sure the Catalog window is open and browse to the Exercise13 folder.**

2 **Right-click the Exercise13 folder and click New > Toolbox.**

3 **Name the new toolbox** Random Tools.tbx**.**



4 **Right-click the new toolbox and click Add > Script.**

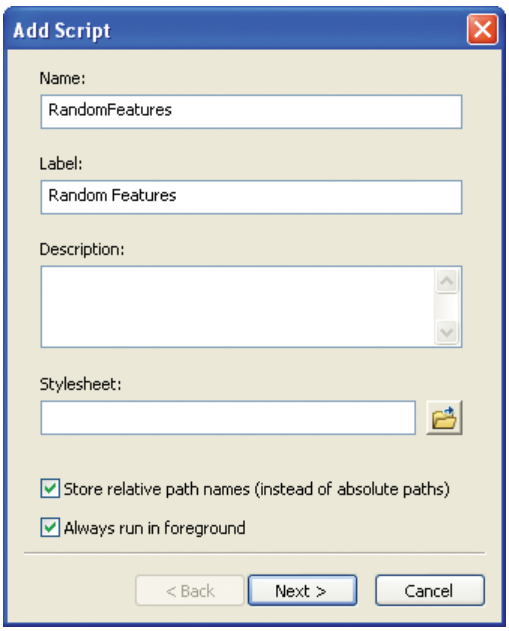
5 **On the Add Script dialog box in the first panel, enter the following information:**

a. For Name, type **RandomFeatures**.

b. For Label, type **Random Features**.

c. Select the check box "Store relative path names".

d. Select the check box "Always run in foreground".



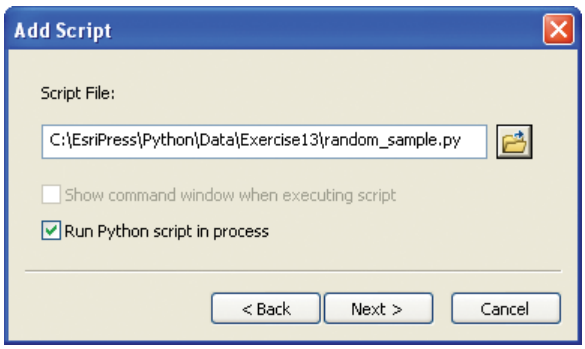
6 **Click Next.**

7 On the Add Script dialog box in the second panel, enter the following information:

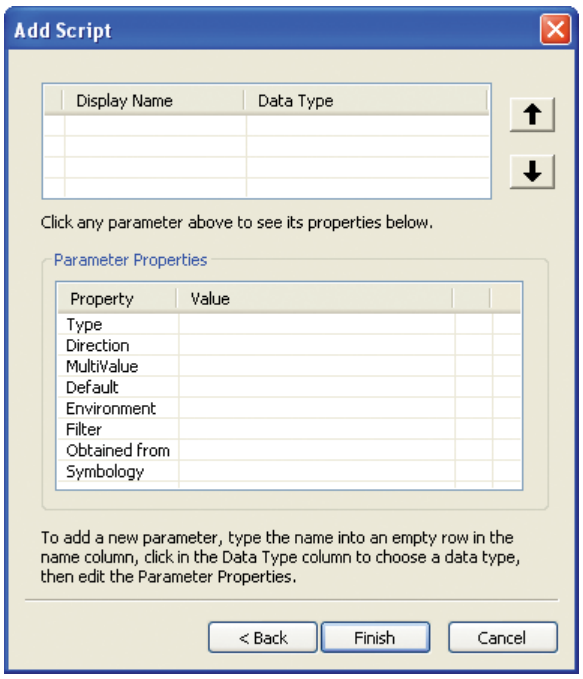
a. For Script File, click the Browse button, navigate to the Exercise13 folder, and double-click the script random_sample.py.

b. Leave the check box "Show command window when executing script" cleared. This check box typically appears shaded.

c. Select the check box "Run Python script in process".

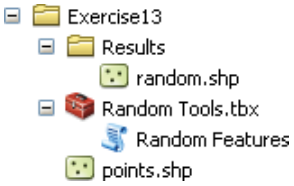


8 Click Next. On the Add Script dialog box in the third panel, you can add the script parameters. For now, you will leave it blank and return to it later.



9 **Click Finish.** The script tool is now created but has no parameters yet, because this step was skipped during the initial creation of the tool.

10 **In the Catalog window, navigate to the Random Tools toolbox. Notice that the Random Features script tool has been added to the toolbox.**



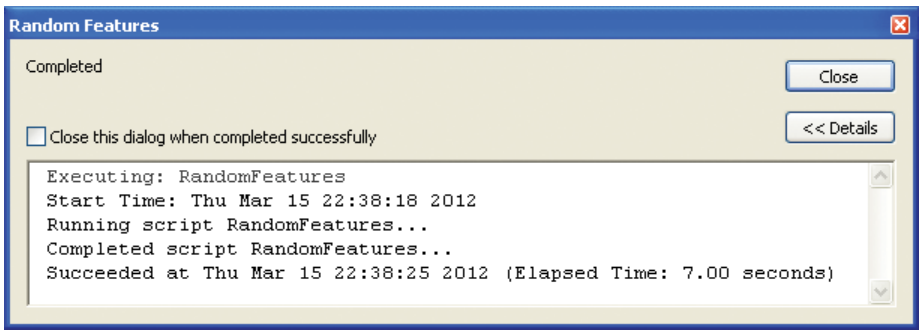
11 **Double-click the Random Features tool.** This opens the dialog box of the newly created tool. The tool does not have any parameters so the dialog box is not very informative.

The tool can be executed, however—that is, the script can be run from the new tool dialog box.

12 **Leave the tool dialog box open. In the ArcMap table of contents, right-click the random layer and click Remove.**

13 **In the Catalog window, browse to the Results folder for exercise 13. Right-click the feature class random.shp and click Delete.**

14 **On the Random Features tool dialog box, click OK.** This runs the script, and the progress dialog box appears.



15 **Click Close to close the tool progress dialog box.**

16 In the Catalog window, browse to the Results folder for exercise 13 and confirm that a new feature class, random.shp**, has been created.**

Running the script tool this way is possible, but the script still uses the original hard-coded values. The next step is to add parameters to the tool and pass them to the script so the user can choose the values.

Set the tool parameters

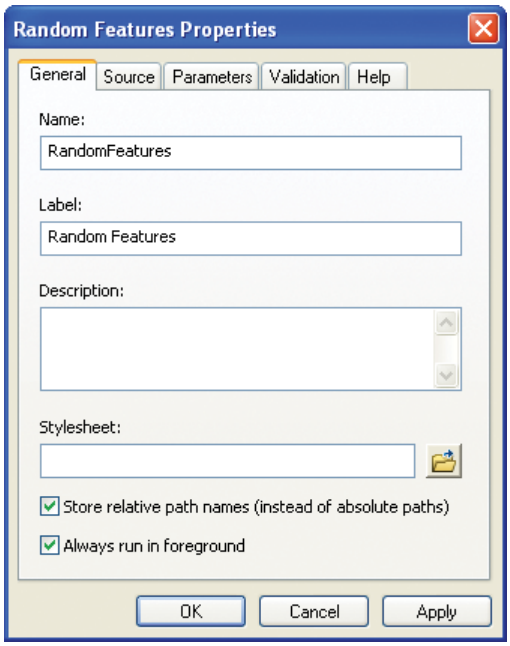
The new tool will have three parameters:

1. An input features class
2. An output feature class containing randomly selected features
3. The number of features to be selected

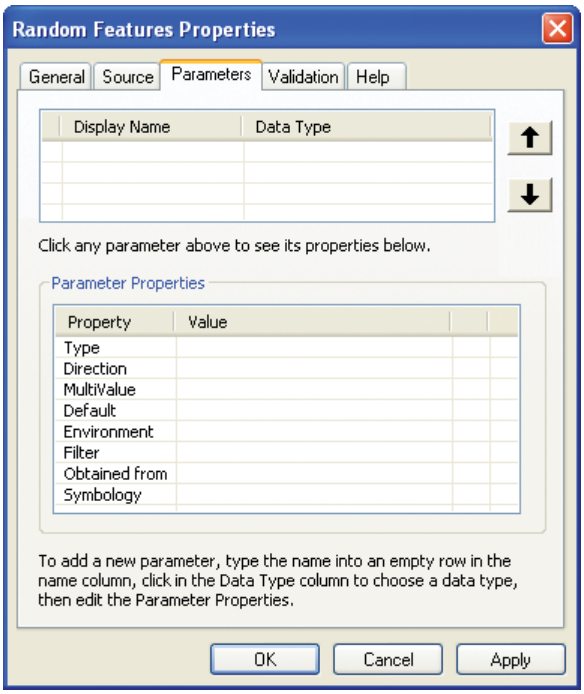
It is possible, of course, to select features based on other criteria, such as a percentage of the input features, but using the number of features will suffice for this exercise.

You will start by setting the tool parameters of the script tool.

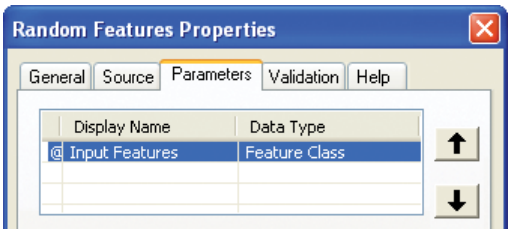
1 In the Catalog window, right-click the Random Features tool and click Properties. This brings up the tool properties dialog box.



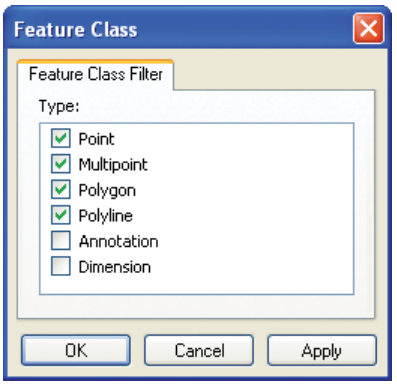
2 Click the Parameters tab. ➔



3 On the tab's uppermost panel, click in the first empty row in the Display Name column and type Input Features**. For Data Type, select Feature Class.** ➔

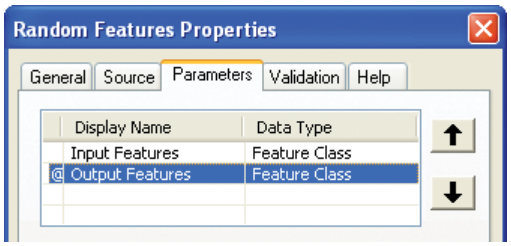


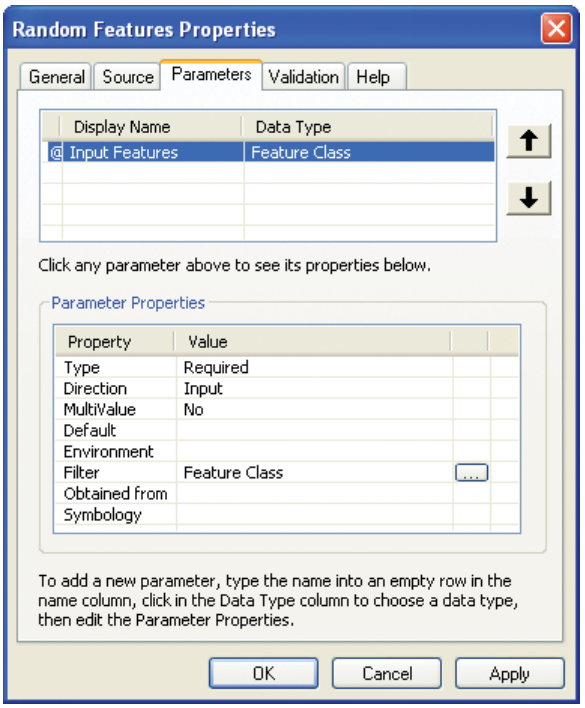
4 In the Parameter Properties panel, click in the cell to the right of the Filter parameter and select Feature Class. On the Feature Class dialog box, clear the Annotation and Dimension check boxes and click OK. ➔



This completes the first parameter. ➔

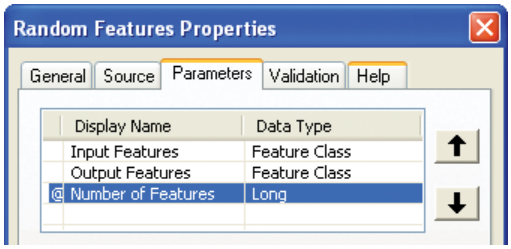
5 Click in the second row in the Display Name column and type Output Features**. For Data Type, select Feature Class.**

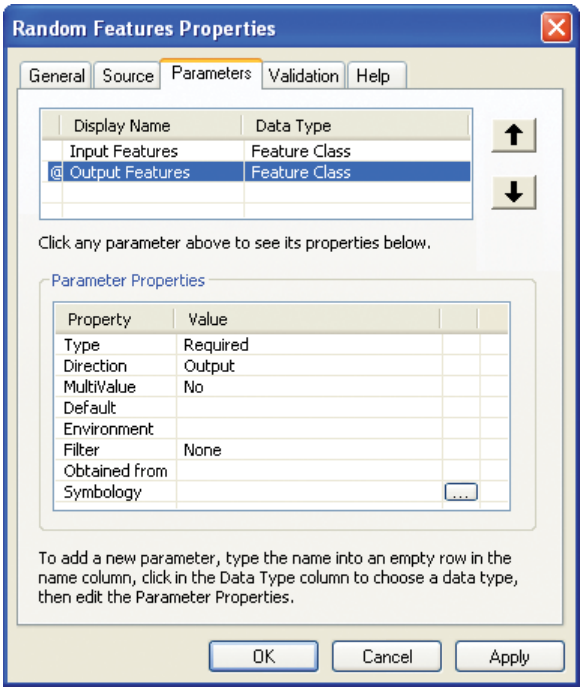




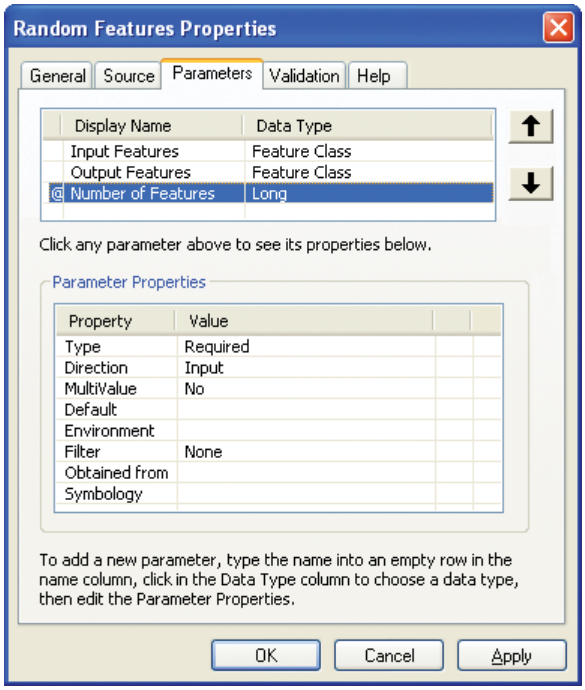
6 In the Parameter Properties panel, click in the cell to the right of the Direction property and select Output. This completes the second parameter. ➔

7 Click in the third row in the Display Name column and type Number of Features**. For Data Type, select Long.**





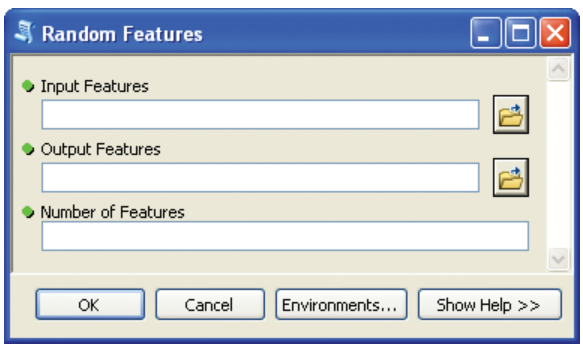
8 **Leave the Parameter Properties at the defaults.** This completes the third parameter.



9 **Click OK.**

Next, you will view the parameters on the tool dialog box.

10 **In the Catalog window, navigate to the Random Tools toolbox. Double-click the Random Features tool.** This opens the tool dialog box with its newly created parameters.

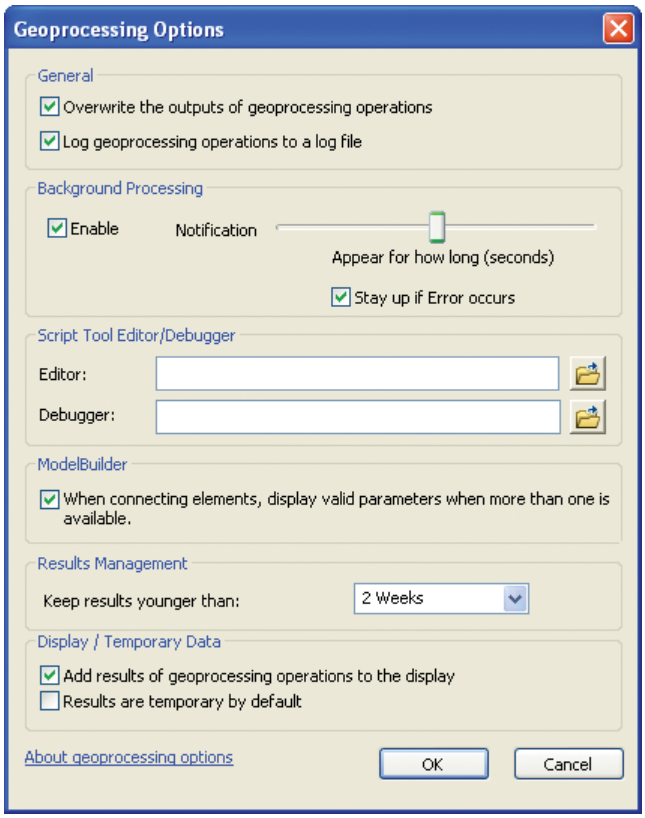


11 **Click Cancel to close the tool dialog box without running the tool.** Now that the parameters are specified, the script needs to be modified to read these parameters.

Select a script editor

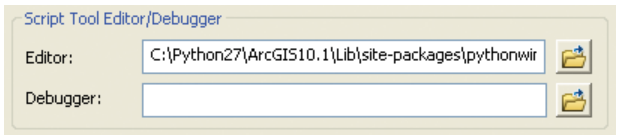
To edit the code of the script, you can open a Python editor and browse to the location of the script file. However, the script can also be opened directly from the script tool. The default editor is IDLE. So, you will first change the default script editor from IDLE to PythonWin.

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



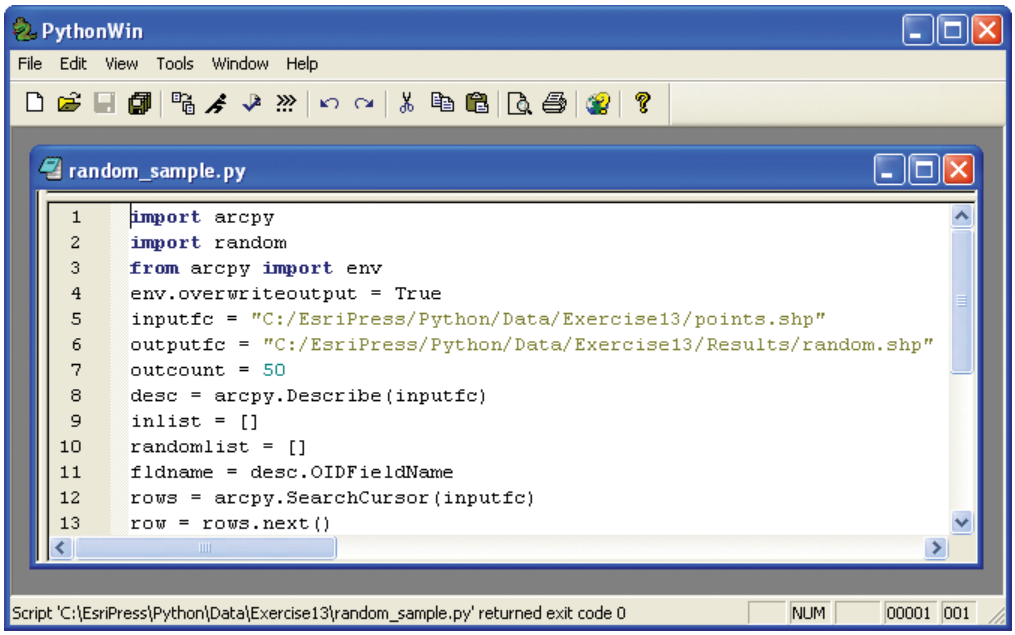
2 In the Script Tool Editor/Debugger panel, click the Browse button to the right of the Editor box.

3 Browse to the location of the PythonWin application. Typically, the path to this application is: C:\Python27\ArcGIS10.1\Lib\site-packages\PythonWin\PythonWin.exe.



4 Click OK to close the Geoprocessing Options dialog box.

5 In the Catalog window, navigate to the Random Tools toolbox. Right-click the Random Features tool and click Edit. This opens the random_sample.py script in PythonWin.



Next, you will make changes to the code, save the script, and run the tool to see if it works correctly.

Edit the tool code to read the parameters

The random_sample.py script needs to be modified to read the parameters from the tool.

1 With the script open in PythonWin, modify lines 5–7 as follows:

```
inputfc = arcpy.GetParameterAsText(0)
outputfc = arcpy.GetParameterAsText(1)
outcount = int(arcpy.GetParameterAsText(2))
```

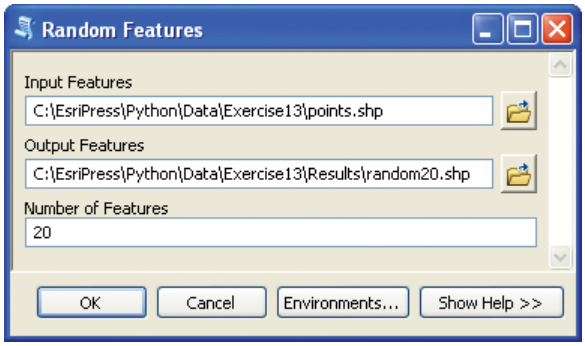
This effectively replaces the hard-coded values in the script with the parameters passed by the tool. The third parameter is received as a string but is converted to an integer.

2 Save the script.

3 Close PythonWin.

4 Return to ArcMap. In the Catalog window, navigate to the Random Tools toolbox. Double-click the Random Features tool.

5 Fill in the tool parameters as shown in the figure.



6 Click OK to run the tool. The tool runs and a new feature class is created with a random selection of 20 point features.

Challenge exercise

Challenge 1

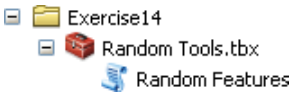
Make a copy of the random_sample.py script and call it **random_percent.py**. Modify the script so that the third parameter is a percentage of the number of input records as an integer between 1 and 100. Modify the script tool settings so that the input for this parameter is validated on the tool dialog box.

Exercise 14
Sharing tools

Prepare the files for sharing tools

In this exercise, you will work with the Random Tools toolbox created in exercise 13 and prepare it for distribution.

1 Start ArcCatalog. Navigate to the C:\EsriPress\Python\Data\Exercise14 folder.

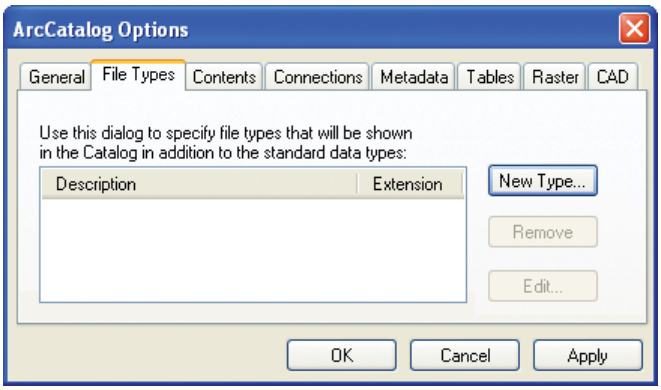


This folder contains the Random Tools toolbox, which contains the finished script tool, Random Features, but without any documentation.

Although by default you can see the toolbox and script tool, Python scripts are not, by default, visible in ArcCatalog. Instead, you can use Windows Explorer (My Computer) to view and manage your scripts. However, you can modify the ArcCatalog options to make Python scripts visible in ArcCatalog. You'll do that next.

2 On the menu bar, click Customize > ArcCatalog Options.

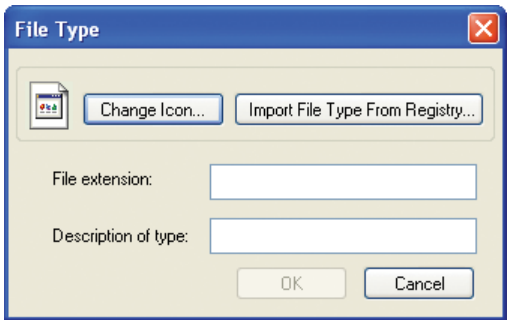
3 On the ArcCatalog Options dialog box, click the File Types tab. ➔



Unless you have previously added file types here, this panel should be empty.

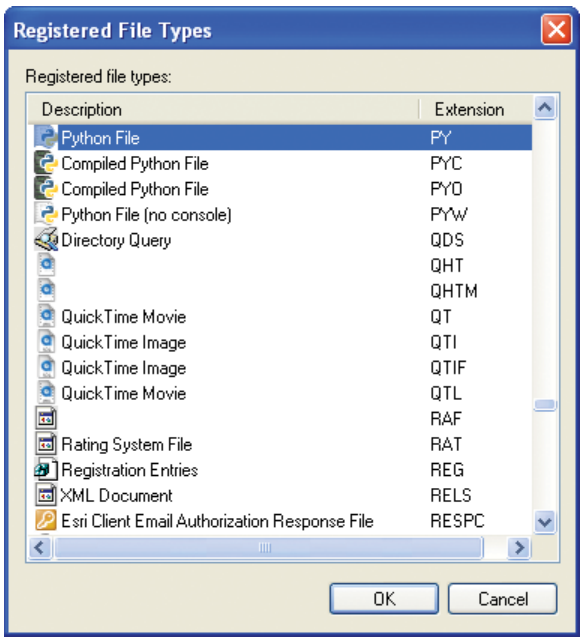
4 Click New Type.

5 On the File Type dialog box, click Import File Type From Registry.

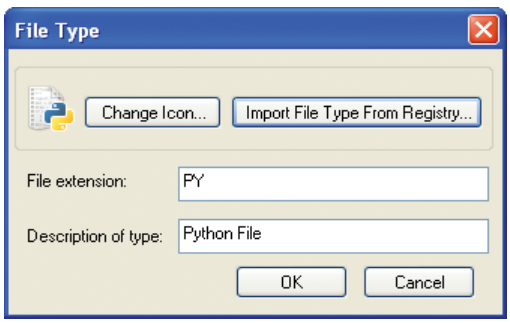


6 On the Registered File Types dialog box, browse to the Python File entry (PY).

Note: File types are sorted based on the file extension, not the description.



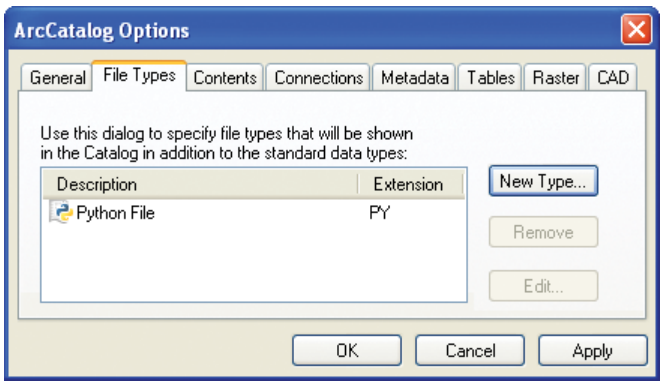
7 Click the Python File entry (PY) and click OK. This brings up the File Type dialog box—this time populated with the details for Python script files, including the icon.



>>> TIP

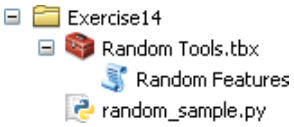
When Windows 7 or Vista is used, depending on the system configuration, only a very limited number of registered file types may be shown. If there is no entry called Python File, the File extension box on the File Type dialog box can be filled in manually. When the File Type dialog box is filled in manually, an icon is not automatically created. An icon can be chosen by clicking the Change Icon button and browsing to an appropriate file. For a typical installation of Python 2.7, the icon can be found at C:\Python27\ArcGIS10.1\DLLs\py.ico.

8 On the File Type dialog box, click OK. Python script files now appear on the File Types tab.



9 On the ArcCatalog Options dialog box, click OK.

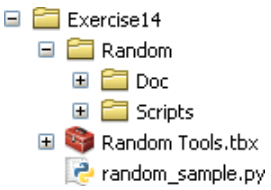
10 In the Catalog tree, right-click the Exercise14 folder and click Refresh. The Python script file now appears in the Catalog tree.



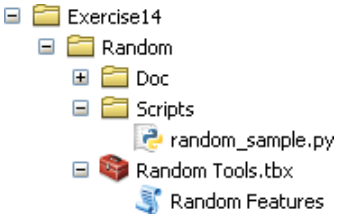
Next, you will organize the files into the standard folder structure.

11 In the Catalog tree, right-click the Exercise14 folder and click New > Folder. Name the folder Random**.**

12 Inside the Random folder, create two new folders called Scripts **and** Doc**.**



13 Move the Random Tools toolbox file into the Random root folder and move the random_sample.py script into the Scripts folder.

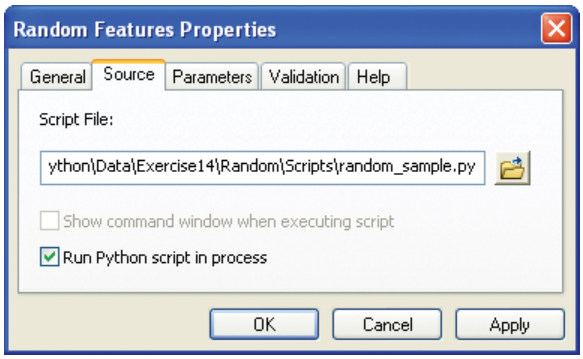


Moving the script relative to the .tbx file means that the relative path needs to be updated, which you'll do next.

14 Right-click the Random Features script tool and click Properties.

15 On the Random Features Properties dialog box, click the Source tab.

16 For Script File, browse to C:\EsriPress\Python\Data\Exercise14\Random\Scripts\random_sample.py.



17 Click OK.

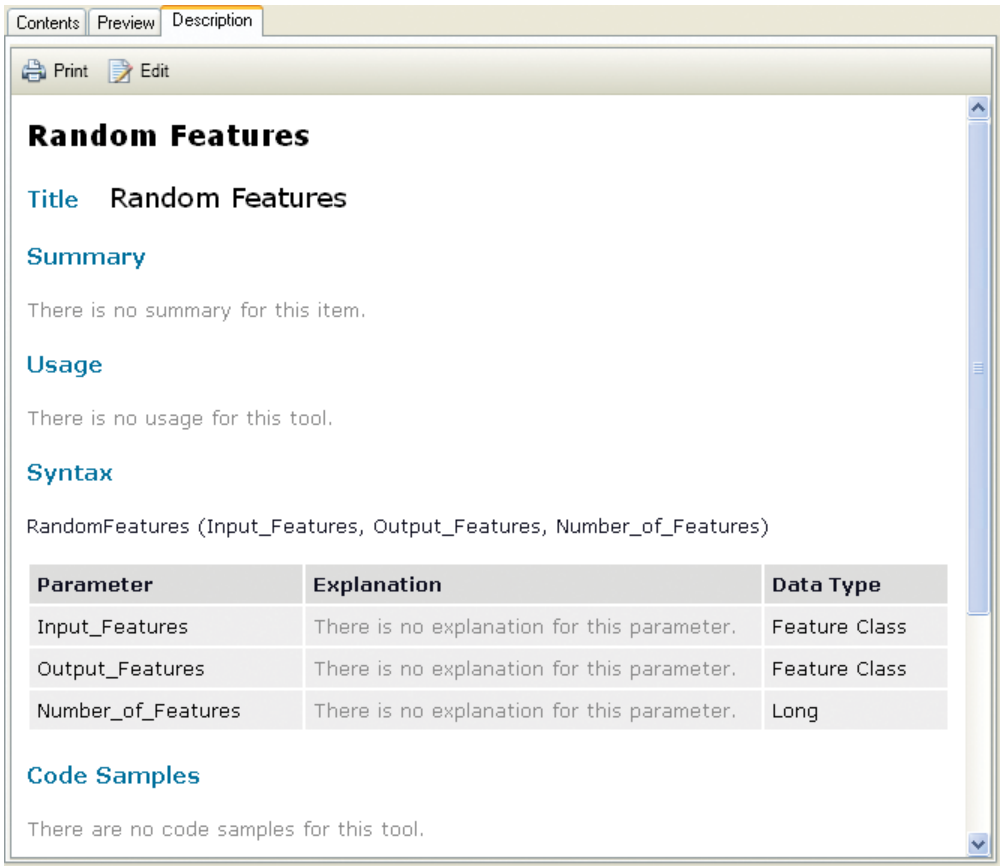
With the folder structure created, additional files can be added, such as sample data or documentation, which you'll add next.

Create tool documentation

When tools are shared, documentation makes them easier to use correctly.

1 In ArcCatalog, in the Catalog tree, click the Random Features script tool.

2 In the panel to the right, click the Description tab.



3 On the panel toolbar under Description, click the Edit button Edit **.**

4 In the Item Description panel, for Summary, type: Creates a new feature class based on a random selection of features in the input feature class.

5 For Usage, type the following as a bullet list:

- The input feature class can be points, polylines, or polygons.
- The output features consist of the same geometry type as the input features.
- The number of features can range from 0 to the number of features in the input feature class.

6 For Syntax, click Input Features > Dialog Explanation and type: Input feature class from which features are to be selected. Can be points, polylines, or polygons.

7 **For Syntax, click Output Features > Dialog Explanation and type:**
Output feature class containing the randomly selected features.

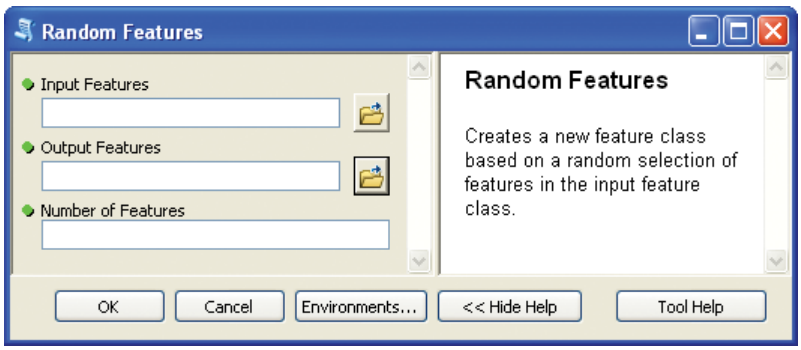
8 **For Syntax, click Number of Features > Dialog Explanation and type:**
The number of features to be selected. Needs to be specified as an integer between 0 and the number of features in the input feature class.
Much more information can be entered on the Description page, but these basic elements illustrate the concept.

9 **On the panel toolbar, click the Save button** Save **.**

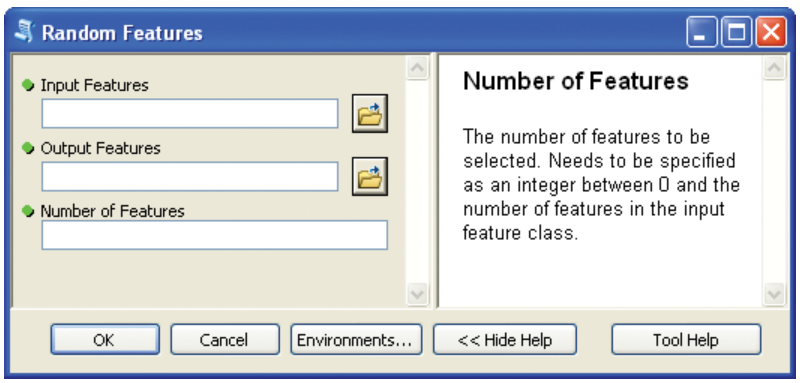
The tool description can now be viewed on the tool dialog box, as you'll see next.

10 **In the Catalog tree, double-click the Random Features script tool.**

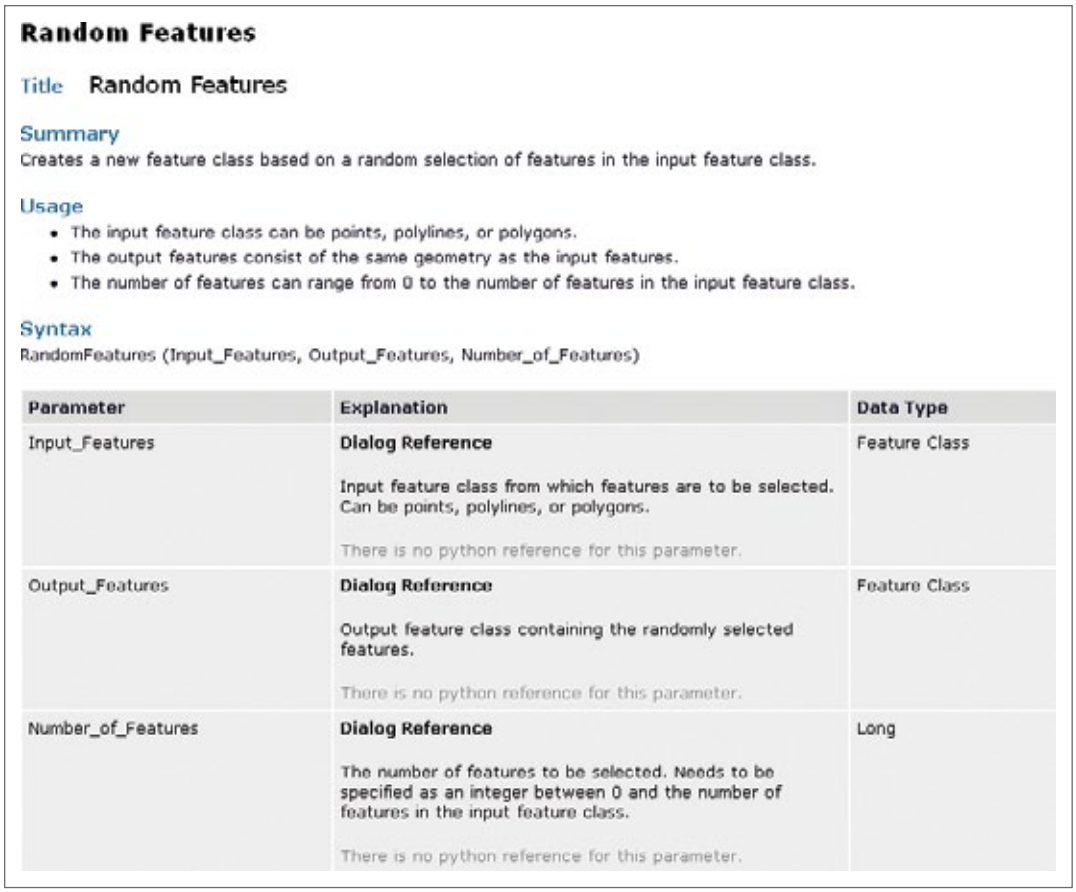
11 **On the Random Features dialog box, click the Show Help button.**
The summary description appears in the Help panel to the right.



12 **On the tool dialog box, click the Input Features parameter to see the description in the Help panel. Also, try the other two parameters, Output Features and Number of Features, to see their Help messages.**



13 Click the Tool Help button to open the Help file in a web browser.



Random Features

Title Random Features

Summary
Creates a new feature class based on a random selection of features in the input feature class.

Usage
- The input feature class can be points, polylines, or polygons.
- The output features consist of the same geometry as the input features.
- The number of features can range from 0 to the number of features in the input feature class.

Syntax
RandomFeatures (Input_Features, Output_Features, Number_of_Features)

Parameter	Explanation	Data Type
Input_Features	**Dialog Reference** Input feature class from which features are to be selected. Can be points, polylines, or polygons. There is no python reference for this parameter.	Feature Class
Output_Features	**Dialog Reference** Output feature class containing the randomly selected features. There is no python reference for this parameter.	Feature Class
Number_of_Features	**Dialog Reference** The number of features to be selected. Needs to be specified as an integer between 0 and the number of features in the input feature class. There is no python reference for this parameter.	Long

14 Scroll down and see your inputs to the tool Description page. Notice that some items may need to be filled in later: Code Samples, Tags, and Credits.

15 Close the browser window.

Challenge exercise

Challenge 1

Go to the Geoprocessing section of the online ArcGIS Resource Center at http://resources.ArcGIS.com/content/geoprocessing. Click the Model and Script Tool Gallery and browse through some of the recent postings. Preview the details of a tool that looks interesting to you. Once you confirm that the tool uses Python, download the tool—if it does not, keep looking until you find a tool that does.

Once the tool is downloaded and extracted, review the contents of the files and answer the following questions:

- How is (are) the tool(s) made available to users? As a .tbx file with one or more .py files? As stand-alone .py files?
- What is the folder structure, if any, of the files that make up the tool?
- What type of documentation is provided, separate from the script itself? Is there a Help page on the tool dialog box?

Open the .py files in PythonWin and review the script's documentation. See if you can recognize the following elements of a script:

- Importing modules, such as ArcPy and others
- Receiving parameters from the tool dialog box, such as `GetParameterAsText`
- Error-handling techniques, such as the `try-except` statement
- Custom functions, such as `def`
- Message handling, such as the `AddMessage` and `AddWarning` functions