Chapter 1
Introduction

Abstract Geospatial data analysis is a key component of decision-making and planning for numerous applications. Geographic Information Systems (GIS), such as ArcGIS®, provide rich analysis and mapping platforms. Modern technology enables us to collect and store massive amounts of geospatial data. The data formats vary widely and analysis requires numerous iterations. These characteristics make computer programming essential for exploring this data. Python is an approachable programming language for automating geospatial data analysis. This chapter discusses the capabilities of scripting for geospatial data analysis, some characteristics of the Python programming language, and the online code and data resources for this book. After downloading and setting up these resources locally, readers can walk through the step-by-step code example that follows. Last, this chapter presents the organization of the remainder of the book.

Chapter Objectives

After reading this chapter, you'll be able to do the following:

- Articulate in general terms, what scripting can do for GIS workflows.
- Explain why Python is selected for GIS programming.
- Install and locate the sample materials provided with the book.
- Contrast the view of compound GIS datasets in Windows Explorer and ArcCatalog™.
- Run code in the ArcGIS® Python Window.

L. Tateosian, *Python For ArcGIS*, DOI 10.1007/978-3-319-18398-5_1

Geographic data analysis involves processing multiple data samples. The analysis may need to be repeated on multiple fields, files, and directories, repeated monthly or even daily, and it may need to be performed by multiple users. Computer programming can be used to automate these repetitive tasks. Scripting can increase productivity and facilitate sharing. Some scriptable tasks involve common data management activities, such as, reformatting data, copying files for backups, and searching database content. Scripts can also harness the tool sets provided by Geographic Information Systems (GIS) for processing geospatial data, i.e., *geoprocessing*. This book focuses on the Python scripting language and geoprocessing with ArcGIS software.

Scripting offers two core capabilities that are needed in nearly any GIS work:

- Efficient batch processing.
- Automated file reading and writing.

Scripts can access or modify GIS datasets and their fields and records and perform analysis at any of these levels. These automated workflows can also be embellished with GUIs and shared for reuse for additional economy of effort.

1.1 Python and GIS

The programming language named Python, created by Guido van Rossum, Dutch computer programmer and fan of the comedy group Monty Python, is an ideal programming language for GIS users for several reasons:

- **Python is easy to pick up**. Python is a nice 'starter' programming language: easy to interpret with a clean visual layout. Python uses English keywords or indentation frequently where other languages use punctuation. Some languages require a lot of set-up code before even creating a program that says 'Hello.' With Python, the code you need to print Hello is `print 'Hello'`.
- **Python is object-oriented**. The idea of object-oriented programming (OOP) was a paradigm shift from functional programming approach used before the 1990s. In functional programming, coding is like writing a collection of mathematical functions. By contrast, object-oriented coding is organized around objects which have properties and functions that do things to that object. OOP languages share common conventions, making it easier for those who have some OOP experience in other languages. There are also advantages to programmers at any level such as context menus which provide cues for completing a line of code.

- **Python help abounds**. Another reason to use Python is the abundance of resources available. Python is an open-source programming language. In the spirit of open-source software, the Python programming community posts plenty of free information online. 'PythonResources.pdf', found with the book's Chapter 1 sample scripts (see Section 1.2), lists some key tutorials, references, and forums.
- **GIS embraces Python**. Due to many of the reasons listed above, the GIS community has adopted the Python programming language. ArcGIS software, in particular has embraced Python and expands the Python functionality with each new release. Python scripts can be used to run ArcGIS geoprocessing tools and more. The term *geoprocessing* refers to manipulating geographic data with a GIS. Examples of geoprocessing include calculating buffer zones around geographic features or intersecting layers of geographic data. The Esri® software, ArcGIS Desktop, even provides a built-in Python command prompt for running Python code statements. The 'ArcGIS Resources' site provides extensive online help for Python, including examples and code templates. Several open-source GIS programs also provide Python programming interfaces. For example, GRASS GIS includes an embedded Python command prompt for running GRASS geoprocessing tools via Python. QGIS and PostGreSQL/PostGIS commands can be also run from Python. Once you know Python for ArcGIS Desktop, you'll have a good foundation to learn Python for other GIS tools.
- **Python comes with ArcGIS**. Python is installed automatically when you install ArcGIS. To work with this book, you need to install ArcGIS Desktop version 10.1 or higher. The example in Section 1.4 shows how to use Python inside of ArcGIS. From Chapter 2 onward, you'll use PythonWin or PyScripter software, instead of ArcGIS, to run Python. Chapter 2 explains the installation procedure for these programs, which only takes a few steps.

1.2 Sample Data and Scripts

The examples and exercises in this book use sample data and scripts available for download from http://www.springer.com/us/book/9783319183978. Click on the 'Supplementary Files' link to download 'gispy.zip'. Place it directly under the 'C:\' drive on your computer. Uncompress the file by right-clicking and selecting 'extract here'. Once this is complete, the resources you need for this book should be inside the 'C:\gispy' directory. Examples and exercises are designed to use data under the 'gispy' directory structured as shown in Figure 1.1.

The download contains sample scripts, a scratch workspace, and sample data:

- **Sample scripts** correspond to the examples that appear in the text. The 'C:\gispy\sample_scripts' directory contains one folder for each chapter. Each time a sample script is referenced by script name, such as 'simpleBuffer.py', it appears in the corresponding directory for that chapter.
- **Scratch workspace** provides a sandbox. 'C:\gispy\scratch' is a directory where output data can be sent. The directory is initially empty. You can run scripts that generate output, check the results in this directory, and then clear this space before starting the next example. This makes it easy to check if the desired output was created and to keep input data directories uncluttered by output files.
- **Sample data** for testing the examples and exercises is located in 'C:\gispy\data'. There is a folder for each chapter. You will learn how to write and run scripts in any directory, but for consistency in the examples and exercises, directories in 'C:\gispy' are specified throughout the book.

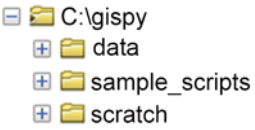

Figure 1.1 Examples in this book use these directories.

1.3 GIS Data Formats

Several GIS data formats are used in this book, including compound data formats such as GRID rasters, geodatabases, and shapefiles. In Windows Explorer, you can see the file components that make up these compound data formats. In ArcCatalog™, which is designed for browsing GIS data formats, you see the software interpretation of these files with both geographic and tabular previews of the data. This section looks at three examples (GRID rasters, Shapefiles, and Geodatabase) comparing the Windows Explorer data representations with the ArcCatalog ones. We will also discuss two additional data types (dBASE and layer files) used in this book that consist of only one file each, but require some explanation.

1.3.1 GRID Raster

A *GRID raster* defines a geographic space with an array of equally sized cells arranged in columns and rows. Unlike other raster formats, such as JPEG or PNG, the file name does have a file extension. The file format consists of two directories, each of which contains multiple files. One directory has the name of the raster and contains '.adf' files which store information about extent, cell resolution, and so

forth; the other directory, named 'info', contains '.dat' and '.nit' files which store file organization information. Figure 1.2 shows a GRID raster named 'getty_rast' in Windows Explorer (left) and in ArcCatalog (right). Windows Explorer, displays the two directories, 'getty_rast' and 'info' that together define the raster named 'getty_rast'. The ArcCatalog directory tree displays the same GRID raster as a single item with a grid-like icon.

1.3.2 Shapefile

A *shapefile* (also called a stand-alone feature class), stores geographic features and their non-geographic attributes. This is a popular format for storing GIS vector data. *Vector data* stores features as sets of points and lines as opposed to rasters which store data in grid cells. The vector features, consisting of geometric primitives (points, lines, or polygons), with associated data attributes stored in a set of supporting files. Though it is referred to as a shapefile, it consists of three or more files, each with different file extensions. The '.shp' (shapefile), '.shx' (header), and '.dbf' (associated database) files are mandatory. You may also have additional files such as '.prj' (projection) and '.lyr' (layer) files. Figure 1.5 shows the shapefile named 'park' in Windows Explorer (which lists multiple files) and ArcCatalog (which displays only a single file). Shapefiles are often referred to with their '.shp' extension in Python scripts.

1.3.3 dBASE Files

One of the shapefile mandatory file types ('.dbf') can also occur as a stand-alone database file. The '.dbf' file format originated with a database management system named 'dBASE'. This format for storing tabular data is referred to as a dBASE file.

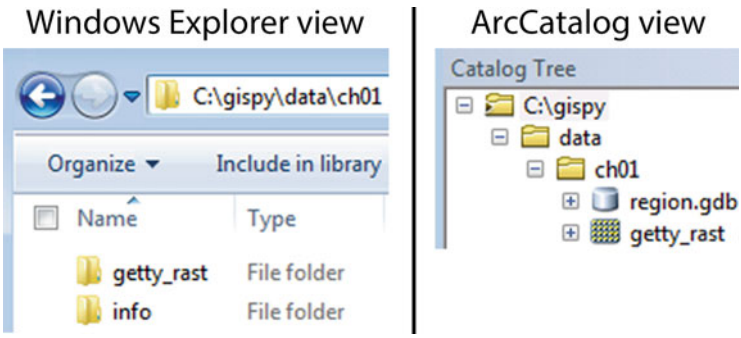

Figure 1.2 Windows Explorer and ArcCatalog views of an Esri GRID raster dataset, 'getty_rast'.

If a dBASE file appears in a directory with a shapefile by the same (base) name, it is associated with the shapefile. If no shapefile in the directory has the same name, it is a stand-alone file.

1.3.4 Layer Files

A '.lyr' file can be used along with a shapefile to store visualization metadata. Usually when a shapefile is viewed in ArcGIS, the *symbology* (the visual representation elements of features, such as color, outline, and so forth) is randomly assigned. Each time it is previewed in ArcCatalog a polygon shapefile might be displayed with a different color, for example. The symbology can be edited in ArcMap® and then a layer file can store these settings. A layer file contains the specifications for the representation of a geographic dataset (a shapefile or raster dataset) and must be stored in same directory as the geographic data. A common source of confusion is another use of the term 'layer'. Data that is added to a map is referred to as a layer (of data). This is not referring to a file, but rather an attribute of the map, data which it displays.

1.3.5 Geodatabase

Esri has three geodatabase formats: file, personal, and ArcSDE™. A *geodatabase* stores a collection of GIS datasets. Multiple formats of data (raster, vector, tabular, and so forth) can be stored together in a geodatabase. Figure 1.3 shows a *file geodatabase*, 'regions.gdb' in Windows Explorer and in ArcCatalog. The left image shows that 'region.gdb' is the name of a directory and inside the directory is a set of files associated with each of the datasets (with extensions .freelist, .gdbindexes, .gdbtable, .gdbtablx, and so forth), only a few of which are shown in Figure 1.3. The ArcCatalog view in Figure 1.3 shows the five datasets (four vector and one raster) in this geodatabase. The geodatabase has a silo-shaped icon. Clicking the geodatabase name expands the tree to show the datasets stored in the geodatabase. The dataset icons vary based on their formats: 'fireStations' is a point vector dataset, 'landCover' and 'workzones' are polygon vector datasets, 'trail' is a polyline dataset, and 'tree' is a raster dataset. The vector format files are referred to as geodatabase *feature classes*, as opposed to shapefiles, which are stand-alone feature classes. Both geodatabases feature classes and stand-alone feature classes store geographic features and their non-geographic attributes.

> **Note** The best way to copy, delete, move, and rename Esri compound data types is to use ArcCatalog or call tools from a Python script.

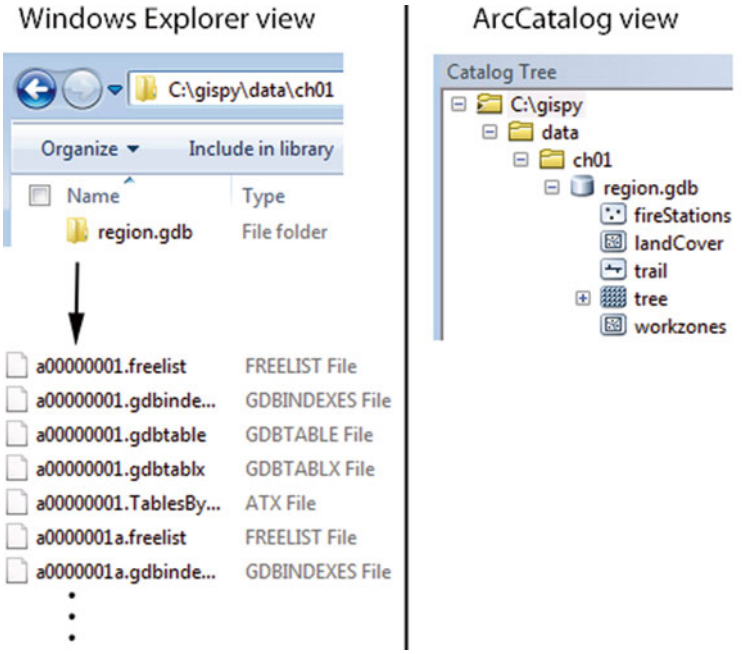

Figure 1.3 Windows Explorer and ArcCatalog views of an Esri file geodatabase, 'region.gdb'.

1.4 An Introductory Example

Example 1.1: This Python script calls the Buffer (Analysis) tool.

```
# simpleBuffer.py
import arcpy
# Buffer park.shp by 0.25 miles. The output buffer erases the
# input features so that the buffer is only outside it.
# The ends of the buffers are rounded and all buffers are
# dissolved together as a single feature.
arcpy.Buffer_analysis('C:/gispy/data/ch01/park.shp',
                      'C:/gispy/scratch/parkBuffer.shp',
                      '0.25 miles', 'OUTSIDE_ONLY', 'ROUND', 'ALL')
```

The ArcGIS Buffer (Analysis) tool, creates polygon buffers around input geographic features (e.g., Figure 1.4). The buffer distance, the side of the input feature to buffer, the shape of the buffer, and so forth can be specified. Buffer analysis has many applications, including highway noise pollution, cell phone tower coverage, and proximity of public parks, to name a few. To get a feel for working with

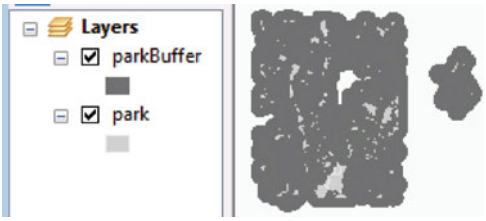

Figure 1.4 Input (*light gray*) and buffer output (*dark gray*) from script Example 1.1 with the input added to the map manually.

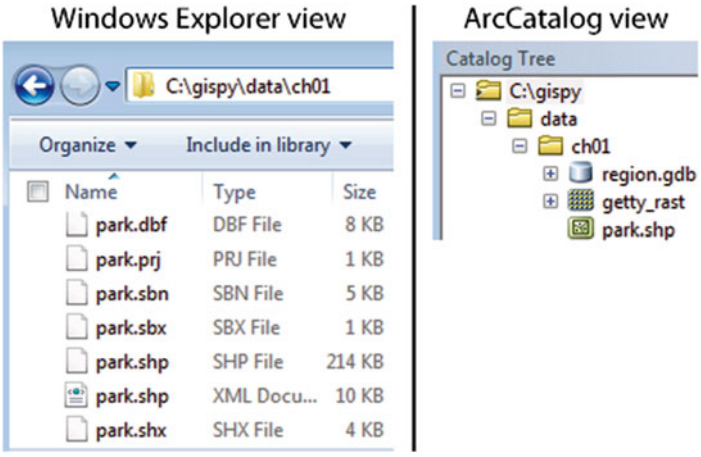

Figure 1.5 Windows Explorer and ArcCatalog views of an Esri shapefile, 'park.shp'.

the sample data and scripts, use a line of Python code to call the Buffer tool and generate buffers around the input features with the following steps:

1. Preview 'park.shp' in 'C:/gispy/data/ch01' using both Windows Explorer and ArcCatalog, as shown in Figure 1.5.
2. When you preview the file in ArcCatalog, a lock file appears in the Windows Explorer directory. Locking interferes with geoprocessing tools. To unlock the file, select another directory in ArcCatalog and then refresh the table of contents (F5). If the lock persists, close ArcCatalog.
3. Open ArcMap. Open the ArcGIS Python Window by clicking the Python button on the standard toolbar, as shown in Figure 1.6 (this window can also be opened from the Geoprocessing Menu under 'Python').
4. Open Notepad (or Wordpad) on your computer.

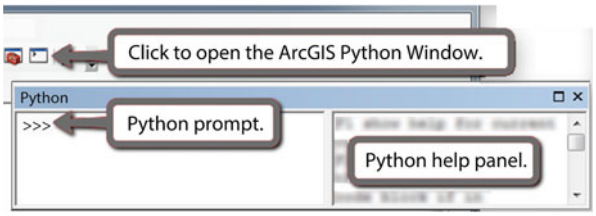

Figure 1.6 The ArcGIS Python Window embedded in ArcMap.

5. In Notepad, browse to the sample scripts for Chapter 1 ('C:\gispy\sample_scripts\ch01') and open 'simpleBuffer.py'. It should look like the script shown in Example 1.1.
6. Copy the last three lines of code from 'simpleBuffer.py' into the ArcGIS Python Window.

Be sure to copy the entirety of all three lines of code, starting with `arcpy.Buffer`, all the way through the right parenthesis. It should look like this:

```
arcpy.Buffer_analysis('C:/gispy/data/ch01/park.shp',
                      'C:/gispy/scratch/parkBuffer.shp',
                      '0.25 miles', 'OUTSIDE_ONLY', 'ROUND', 'ALL')
```

7. Press the 'Enter' key and you'll see messages that the buffering is occurring.
8. When the process completes, confirm that an output buffer file has been created and added to the map (Figure 1.4 displays the output that was automatically added to the map in dark gray and the input which was added to the map by hand in light gray). The feature color is randomly assigned, so your buffer color may be different.
9. Confirm that you see a message in the Python Window giving the name of the result file. If you get an error message instead, then the input data may have been moved or corrupted or the Python statement wasn't copied correctly.
10. You have just called a tool from Python. This is just like running it from the ArcToolbox™ GUI such as the one shown in Figure 1.7 (You can launch this dialog with ArcToolbox > Analysis tools > Proximity > Buffer). The items in the parentheses in the Python tool call are the parameter values, the user input. Compare these values to the user input in Figure 1.7. Can you spot three differences between the parameters used in the Python statement you ran and those used in Figure 1.7?

The ArcMap Python Window is good for running simple code and testing code related to maps (we'll use it again in Chapter 24); However, Chapter 2 introduces other software which we'll use much more frequently to save and run scripts. Before moving on to the organization of the book, let's answer the question posed in step 10. The three parameter value differences are the output file names (C:/gispy/scratch/parkBuffer.shp versus C:\gispy\data\ch01\park_Buff.shp), the buffer distances (0.25 miles versus 5 miles), and the dissolve types (ALL versus NONE).

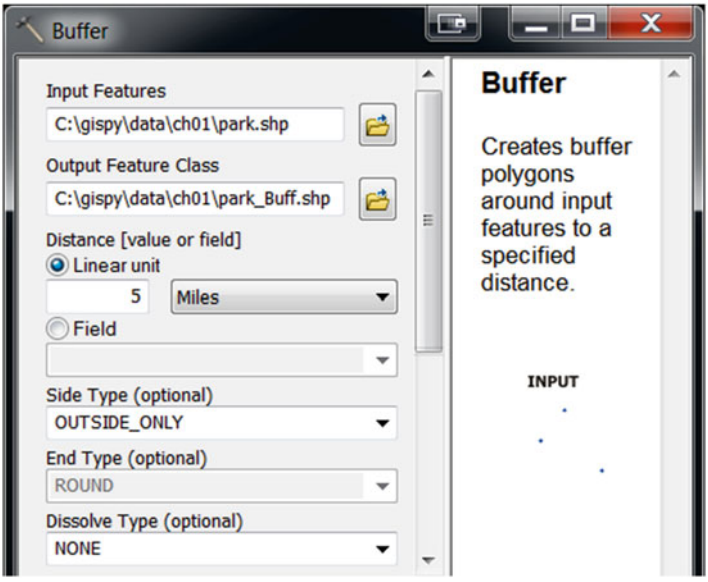

Figure 1.7 Each input slot in the graphical user interface (GUI) for running the buffer tool corresponds to a parameter in the Python code blueprint.

1.5 Organization of This Book

This book focuses on automatically reading, writing, analyzing, and mapping geospatial data. The reader will learn how to create Python scripts for repetitive processes and design flexible, reusable, portable, robust GIS processing tools. The book uses Python to work with the ArcGIS arcpy™ package, as well as HTML, KML, and SQL. Script tool user-interfaces, Python toolboxes, and the arcpy mapping module are also discussed. Expositions are accompanied by interactive code snippets and full script examples. Readers can run the interactive code examples while reading along. The script examples are available as Python files in the chapter 'sample scripts' directory downloadable from the Springer Web site (as explained in Section 1.2). Each chapter begins with a set of learning objectives and concludes with a list of 'key terms' and a set of exercises.

The chapters are designed to be read sequentially. General Python concepts are organized to build the Python skills needed for the ArcGIS scripting capabilities. General Python topic discussions are presented with GIS examples. Chapter 2 introduces the Python programming language and the software used to run Python scripts. Chapters 3 and 4 discuss four core Python data types. Chapters 5 and 6 cover ArcGIS tool help and calling tools with Python using the arcpy package. Chapter 7 deals with getting input from the user. Chapter 8 introduces programming control structures. This provides a backdrop for the decision-making and looping

syntax discussed in Chapters 9 and 10. Python can use special arcpy functions to describe data (used for decision-making in Chapter 9) and list GIS data. Batch geoprocessing is performed on lists of GIS data in Chapter 11. Chapter 12 highlights some additional useful list manipulation techniques.

As scripts become more complex, debugging (Chapter 13) and handling errors (Chapter 14) become important skills. Creating reusable code by defining functions (Chapter 15) and modules (Chapter 16) also becomes key. Next, Chapter 17 discusses reading and writing data records using arcpy cursors. In Chapter 18, another Python data structure, called a dictionary, is introduced. Dictionaries can be useful during file reading and writing, which is discussed in Chapter 19. Chapter 20 explains how to access online data, decompress files, and read, write, and parse markup languages. Another code reuse technique, the user-defined class, is presented in Chapter 21. Chapters 22 and 23 show how to create GUIs for file input or other GIS data types with Script Tools and Python toolboxes. Finally, Chapter 24 uses the arcpy mapping module to perform mapping tasks with Python.

1.6 Key Terms

Geoprocessing
GRID rasters
Vector data
Symbology
dBASE file
Layer file
Geodatabase
Feature class
Window Explorer vs. ArcCatalog
Buffer (Analysis) tool
ArcGIS Python Window