

Laura Tateosian

Python For ArcGIS

Python For ArcGIS

Laura Tateosian

Python For ArcGIS



Springer

Laura Tateosian
North Carolina State University
Raleigh, NC, USA

ISBN 978-3-319-18397-8 ISBN 978-3-319-18398-5 (eBook)
DOI 10.1007/978-3-319-18398-5

Library of Congress Control Number: 2015943490

Springer Cham Heidelberg New York Dordrecht London
© Springer International Publishing Switzerland 2015

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, express or implied, with respect to the material contained herein or for any errors or omissions that may have been made.

Esri images are used by permission. Copyright © 2015 Esri. All rights reserved.

Python is copyright the Python Software Foundation. Used by permission.

PyScripter is an OpenSource software authored by Kiriakos Vlahos.

Printed on acid-free paper

Springer International Publishing AG Switzerland is part of Springer Science+Business Media
(www.springer.com)

Preface



You know... this would be a lot faster with a script!

Imagine... You've just begun a new job as a GIS specialist for the National Park Service. Your supervisor has asked you to analyze some wildlife data. She gives you a specific example to start with: One data table (Bird Species) contains a list of over 600 bird species known to be native to North Carolina. Another data table (Bird Inventory) contains over 5,000 records, each corresponding to a sighting of a particular bird. Your task is to clean the data, reformat the file for GIS compatibility, and summarize the data by determining what percent of the native Bird Species appear in the inventory and map the results. Once you complete this, she would like

you to repeat this process for historical datasets for the last 10 years of monthly records. After that, the next assignment will be to answer the same question based on monthly species inventory datasets for fish and invertebrates.

Performing this process manually for one dataset could be time consuming and error prone. Performing this manually for numerous datasets is completely impractical. Common GIS tasks such as this provide a strong motivation for learning how to automate workflows.

Python programming to the rescue! This analysis can be performed in single Python script. You could even use Python to automatically add the sighting locations to ArcMap® and create a report with the results. You could also add a graphical user interface (GUI) so that colleagues can run the analysis themselves on other datasets. This book will equip you to do all of these things and more.

This book is aimed primarily at teaching GIS Python scripting to GIS users who have little or no Python programming experience. Topics are organized to build upon each other, so for this audience, reading the chapters sequentially is recommended. Computer programmers who are not familiar with ArcGIS can also use this book to learn to automate GIS analyses with the ArcGIS Python API, accessible through the `arcpy` package. These readers can quickly peruse the general Python concepts and focus mainly on the sections describing `arcpy` package functionality.

The book comes with hundreds of example Python scripts. Download these along with the sample datasets so that you can run the examples and modify them to experiment as you follow the text. Where and how to download the supplementary material is explained in the first chapter. Rolling your sleeves up and trying things yourself will really boost your understanding as you use this book. Remember, the goal is to become empowered to save time by using Python and be more efficient in your work. Enjoy!

Raleigh, NC

Laura Tateosian

Acknowledgements

Thank you to the numerous people who have helped with this project. Over the years, many students have studied from drafts of this material and made suggestions for improvements. They have also expressed enthusiasm about how useful this knowledge has been for their work, which encouraged me to enable a wider distribution by writing this book.

Special thanks to those who have reviewed the book, Pankaj Chopra of Emory University, Brian McLean of the North Carolina Department of Agriculture and Consumer Services, Holly Brackett of Wake Technical Community College, Rahul Bhosle of GIS Data Resources, Inc., and Makiko Shukunobe of North Carolina State University. Thank you to Sarah Tateosian for the artwork. Thank you to Tom Danninger, Michael Kanders, Joe Roise Justin Shedd, and Bill Slocumb of North Carolina State University and Cheryl Sams at the National Park Service for assistance with datasets. I also would like to thank Mark Hammond and Kiriakos Vlahos, the authors of PythonWin and PyScripter, respectively. Finally, I would like to express my gratitude for mentorship from Hugh Devine, Christopher Healey, Helena Mitasova, Sarah Stein, and Alan Tharp.

Contents

1	Introduction.....	1
1.1	Python and GIS	2
1.2	Sample Data and Scripts	3
1.3	GIS Data Formats	4
1.3.1	GRID Raster.....	4
1.3.2	Shapefile.....	5
1.3.3	dBASE Files.....	5
1.3.4	Layer Files	6
1.3.5	Geodatabase	6
1.4	An Introductory Example	7
1.5	Organization of This Book.....	10
1.6	Key Terms	11
2	Beginning Python.....	13
2.1	Where to Write Code	13
2.2	How to Run Code in PythonWin and PyScripter.....	15
2.3	How to Pass Input to a Script.....	20
2.4	Python Components	21
2.4.1	Comments	23
2.4.2	Keywords	23
2.4.3	Indentation	24
2.4.4	Built-in Functions	24
2.4.5	Variables, Assignment Statements, and Dynamic Typing.....	26
2.4.6	Variables Names and Tracebacks.....	28
2.4.7	Built-in Constants and Exceptions.....	30
2.4.8	Standard (Built-in) Modules	31
2.5	Key Terms	32
2.6	Exercises	33
3	Basic Data Types: Numbers and Strings.....	37
3.1	Numbers.....	37
3.2	What Is a String?.....	38

3.3	String Operations	40
3.3.1	Find the Length of Strings	40
3.3.2	Indexing into Strings.....	41
3.3.3	Slice Strings	42
3.3.4	Concatenate Strings	43
3.3.5	Check for Substring Membership	44
3.4	More Things with Strings (a.k.a. String Methods)	45
3.5	File Paths and Raw Strings	49
3.6	Unicode Strings.....	51
3.7	Printing Strings and Numbers.....	52
3.8	Key Terms	54
3.9	Exercises	54
4	Basic Data Types: Lists and Tuples	59
4.1	Lists.....	59
4.1.1	Sequence Operations on Lists.....	60
4.1.2	List Methods	61
4.1.3	The Built-in <code>range</code> Function.....	62
4.1.4	Copying a List.....	62
4.2	Tuples.....	64
4.3	Syntax Check and Tracebacks	65
4.4	Key Terms	69
4.5	Exercises	70
5	ArcGIS and Python.....	77
5.1	ArcToolbox	77
5.2	ArcGIS Python Resources	79
5.3	Exporting Models.....	81
5.4	Working with GIS Data.....	83
5.5	ArcGIS + Python = arcpy	84
5.6	arcpy Functions.....	87
5.7	Environment Settings.....	89
5.8	Key Terms	92
5.9	Exercises	92
6	Calling Tools with Arcpy	95
6.1	Calling Tools	95
6.2	Help Resources	97
6.2.1	Tool Help	97
6.2.2	Code Snippets	98
6.3	Tool Parameters.....	99
6.3.1	Linear Units	99
6.3.2	Python Expressions as Inputs.....	100
6.3.3	Multivalue Inputs	101
6.3.4	Optional Parameters.....	102
6.4	Return Values and Result Objects.....	103

6.5	Spatial Analyst Toolbox.....	107
6.5.1	Calling Spatial Analyst tools.....	107
6.5.2	Importing spatial analyst.....	109
6.5.3	Raster Calculator.....	109
6.6	Temporary Feature Layers	111
6.7	Using Variables for Multiple Tool Calls	111
6.8	Calling Custom Tools.....	114
6.9	A Word About Old Scripts.....	115
6.10	Discussion.....	115
6.11	Key Terms	116
6.12	Exercises	116
7	Getting User Input	119
7.1	Hard-coding versus Soft-coding	119
7.2	Using GetParameterAsText.....	120
7.3	Using sys.argv.....	121
7.4	Missing Arguments	122
7.5	Argument Spacing.....	123
7.6	Handling File Names and Paths with os Module Functions.....	125
7.6.1	Getting the Script Path	128
7.7	Key Terms	129
7.8	Exercises	129
8	Controlling Flow	133
8.1	Outlining Workflow	133
8.2	Key Terms	139
8.3	Exercises	139
9	Decision-Making and Describing Data	141
9.1	Conditional Expressions	145
9.1.1	Comparison Operators	146
9.1.2	Equality vs. Assignment.....	148
9.1.3	Logical Operators.....	148
9.1.4	Membership Operators.....	149
9.2	ArcGIS Tools That Make Selections.....	150
9.2.1	Select by Attributes and Temporary Feature Layers.....	153
9.3	Getting a Description of the Data	154
9.3.1	Describe Object Properties.....	155
9.3.2	Lists of Properties.....	156
9.3.3	Using Specialized Properties.....	156
9.3.4	Compound vs. Nested Conditions.....	158
9.3.5	Testing Conditions.....	160
9.4	Required and Optional Script Input	161
9.5	Creating Output Directories	165
9.6	Key Terms	166
9.7	Exercises	167

10 Repetition: Looping for Geoprocessing	171
10.1 Looping Syntax	171
10.1.1 WHILE-Loops	172
10.1.2 FOR-Loops	175
10.2 Nested Code Blocks.....	179
10.3 Directory Inventory.....	180
10.4 Indentation and the TabNanny	183
10.5 Key Terms	184
10.6 Exercises	184
11 Batch Geoprocessing.....	187
11.1 List GIS Data	187
11.2 Specify Data Name and Type.....	190
11.3 List Fields	194
11.4 Administrative Lists	196
11.5 Batch Geoprocess Lists of Data.....	197
11.6 Debug: Step Through Code	200
11.7 Key Terms	203
11.8 Exercises	204
12 Additional Looping Functions	209
12.1 List Comprehension.....	209
12.2 The Built-in enumerate Function.....	211
12.3 The Built-in zip Function	213
12.4 Walking Through Subdirectories	214
12.5 Key Terms	219
12.6 Exercises	220
13 Debugging	223
13.1 Syntax Errors	224
13.2 Exceptions.....	224
13.3 Logic Errors	226
13.4 PythonWin Debugging Toolbar	229
13.4.1 Using Breakpoints.....	230
13.5 Running in Debug Mode.....	234
13.6 PyScripter Debugging Toolbar.....	235
13.7 Debugging Tips.....	237
13.8 Key Terms	237
13.9 Exercises	237
14 Error Handling.....	241
14.1 try/except Structures	243
14.1.1 Using Named Exceptions	244
14.1.2 Multiple except Blocks	245
14.1.3 Error Handling Gotcha	247
14.2 Geoprocessing and Error Handling.....	248
14.2.1 Getting Geoprocessing Messages	249
14.2.2 The arcpy Named Exception.....	251
14.3 Catching Exceptions in Loops	252

14.4	Discussion.....	255
14.5	Key Terms.....	256
14.6	Exercises	257
15	User-Defined Functions	261
15.1	A Word About Function Words	261
15.1.1	How It Works	263
15.1.2	The Docstring.....	265
15.2	Custom Functions with Arguments	266
15.2.1	Script Arguments vs. Functions Arguments	268
15.2.2	Optional Arguments	270
15.3	Returning Values.....	271
15.3.1	A Common Mistake: Where Did the None Come from?.....	274
15.3.2	Returning Multiple Values	276
15.4	When to Write Functions.....	277
15.4.1	Where to Define Functions.....	279
15.5	Variables Inside and Outside of Functions	280
15.5.1	Mutable Arguments Can Change	280
15.5.2	Pass in Outside Variables	282
15.6	Key Terms.....	283
15.7	Exercises	283
16	User-Defined Modules	291
16.1	Importing User-Defined Modules.....	291
16.2	Using Functions in Another Module	296
16.3	Modifying User-Defined Modules (Reload!)	298
16.4	Am I the Main Module? What's My Name?	299
16.5	Time Handling Example.....	301
16.6	Summary	304
16.7	Key Terms.....	304
16.8	Exercises	304
17	Reading and Writing with Cursors	309
17.1	Introduction to Cursors	310
17.2	Reading Rows.....	311
17.3	The Field Names Parameter.....	313
17.4	The Shape Field and Geometry Tokens.....	315
17.5	Looping with Cursors	317
17.6	Locking	317
17.6.1	The del Statement.....	319
17.6.2	The with Statement	320
17.7	Update Cursors	321
17.8	Insert Cursors.....	322
17.8.1	Inserting Geometric Objects	325
17.9	Selecting Rows with SQL.....	328
17.10	Key Terms	330
17.11	Exercises	331

18	Dictionaries.....	335
18.1	Dictionary Terms and Syntax.....	336
18.1.1	Access by Key, Not by Index.....	337
18.1.2	Conditional Construct vs. Dictionary	338
18.1.3	How to Modify: Update/Add/Delete Items	339
18.2	Dictionary Operations and Methods.....	341
18.2.1	Does It Have That Key?.....	341
18.2.2	Listing Keys, Values, and Items.....	342
18.2.3	Looping Through Dictionaries	343
18.3	Populating a Dictionary	344
18.3.1	Dictionaries and Cursors.....	348
18.4	Discussion.....	350
18.5	Key Terms	350
18.6	Exercises	350
19	Reading and Writing Text Files.....	357
19.1	Working with <code>file</code> Objects.....	357
19.1.1	The <code>WITH</code> Statement.....	359
19.1.2	Writing Text Files	359
19.1.3	Safe File Reading	362
19.1.4	The <code>os</code> Working Directory vs. the <code>arcpy</code> workspace.....	362
19.1.5	Reading Text Files	364
19.2	Parsing Line Contents	365
19.2.1	Parsing Field Names	369
19.3	Modifying Text Files.....	370
19.3.1	Pseudocode for Modifying Text Files.....	372
19.3.2	Working with Tabular Text	372
19.4	Pickling	376
19.5	Discussion.....	378
19.6	Key Terms	379
19.7	Exercises	379
20	Working with HTML and KML.....	385
20.1	Working with HTML	385
20.1.1	Specifying Links.....	387
20.1.2	Embedding Images	388
20.1.3	HTML Lists	389
20.1.4	HTML Tables.....	389
20.1.5	Writing HTML with Python	390
20.1.6	Parsing HTML with BeautifulSoup.....	393
20.2	Fetching and Uncompressing Data	397
20.2.1	Fetching HTML	397
20.2.2	Fetching Compressed Data	398
20.2.3	Expanding Compressed Data.....	399
20.3	Working with KML.....	401
20.3.1	The Structure of KML	402

20.3.2	Parsing KML.....	403
20.3.3	Converting KML to Shapefile.....	405
20.4	Discussion	406
20.5	Key Terms	407
20.6	Exercises	407
21	Classes	415
21.1	Why Use OOP?.....	416
21.2	Defining a Class	418
21.3	Object Initialization and Self	419
21.4	Using Class Objects	420
21.5	Where to Define a Class.....	425
21.6	Classes Within a Separate User-Defined Module	427
21.7	Discussion	428
21.8	Key Terms	430
21.9	Exercises	430
22	User Interfaces for File and Folder Selection	435
22.1	A Simple Interface with <code>raw_input</code>	435
22.2	File Handling with <code>tkFileDialog</code>	436
22.2.1	Getting File and Directory Names.....	437
22.2.2	Options.....	439
22.2.3	Opening Files for Reading and Writing.....	443
22.3	Discussion	444
22.4	Key Terms	445
22.5	Exercises	446
23	ArcGIS Python GUIs	449
23.1	Creating a Script Tool	450
23.1.1	Printing from a Script Tool	452
23.1.2	Making a Script Tool Button	454
23.1.3	Pointing to a Script	456
23.2	Creating a GUI.....	458
23.2.1	Using Parameter Data Types.....	461
23.2.2	Using Parameter Properties	464
23.3	Showing Progress.....	478
23.4	Validating Input.....	481
23.4.1	The <code>ToolValidator</code> Class.....	483
23.5	Python Toolboxes.....	489
23.5.1	Setting Up Parameters (<code>getParameterInfo</code>)	491
23.5.2	Checking for Licenses (<code>isLicensed</code>).....	492
23.5.3	Validation (<code>updateParameters</code> and <code>updateMessages</code>).....	493
23.5.4	Running the Code (<code>execute</code>).....	494
23.5.5	Comparing Tools.....	496
23.6	Discussion	499
23.7	Key Terms	500
23.8	Exercises	500

24	Mapping Module.....	505
24.1	Map Documents.....	507
24.1.1	Map Name or 'CURRENT' Map	510
24.1.2	MapDocument Properties.....	511
24.1.3	Saving Map Documents.....	513
24.2	Working with Data Frames	514
24.3	Working with Layers.....	515
24.3.1	Moving, Removing, and Adding Layers	517
24.3.2	Working with Symbology	520
24.4	Managing Layout Elements	522
24.5	Discussion.....	525
24.6	Key Terms	525
24.7	Exercises	526
	Index.....	531

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.

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.

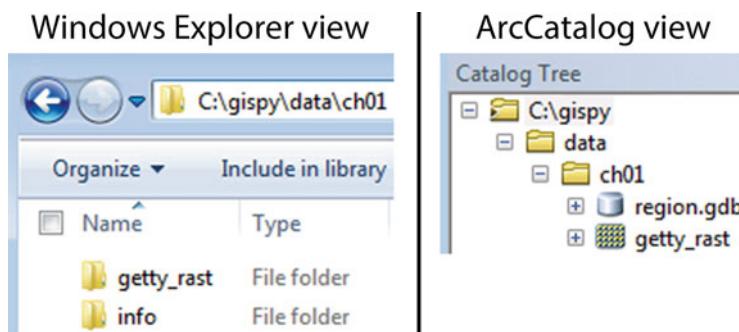


Figure 1.2 Windows Explorer and ArcCatalog views of an Esri GRID raster dataset, ‘getty_rast’.

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.

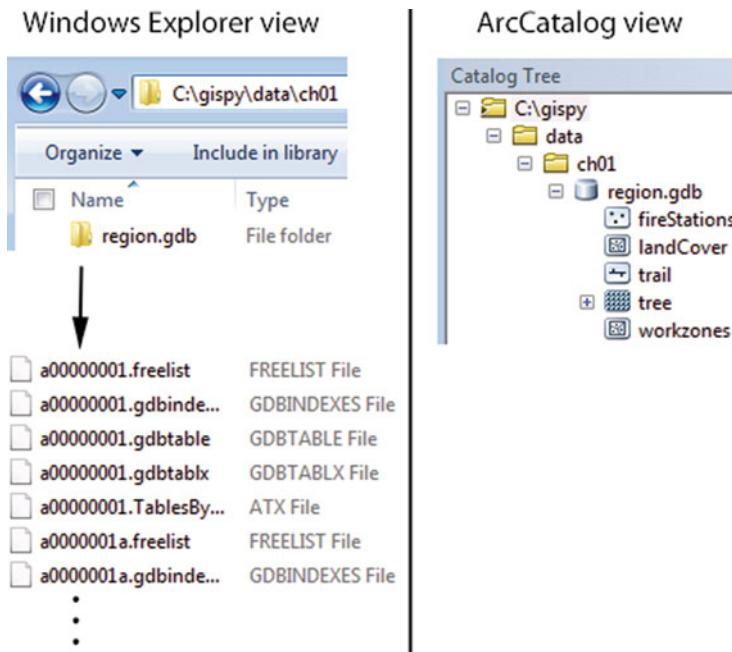


Figure 1.3 Windows Explorer and ArcCatalog views of an Esri file geodatabase, ‘region.gdb’.

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.

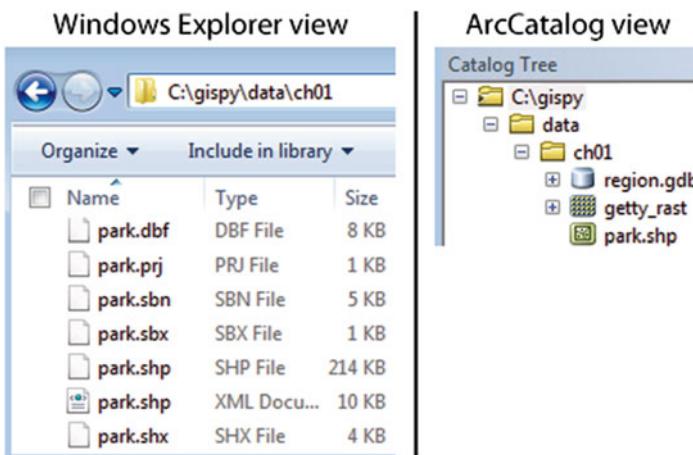


Figure 1.5 Windows Explorer and ArcCatalog views of an Esri shapefile, ‘park.shp’.

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.

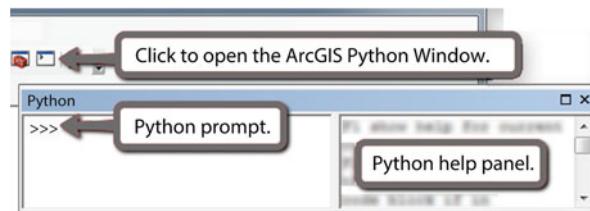


Figure 1.6 The ArcGIS Python Window embedded in ArcMap.

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).

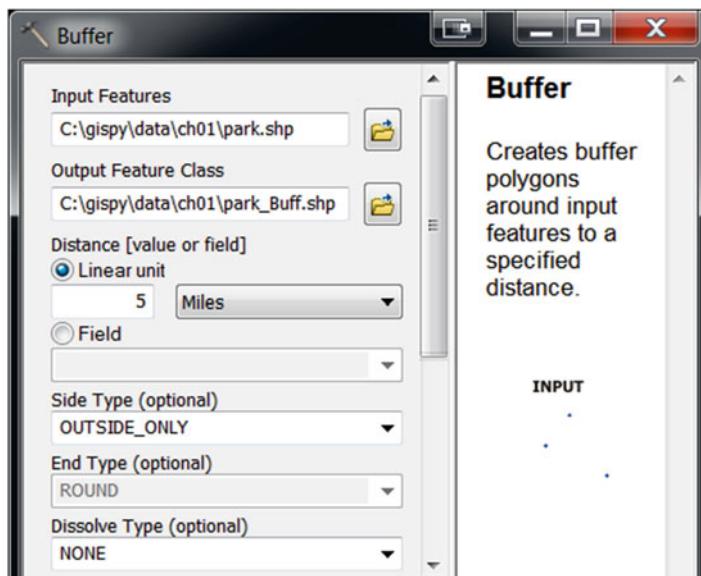


Figure 1.7 Each input slot in the graphical user interface (GUI) for running the buffer tool corresponds to a parameter in the Python code blueprint.

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