



Quick answers to common problems

# Programming ArcGIS 10.1 with Python Cookbook

Over 75 recipes to help you automate geoprocessing tasks,  
create solutions, and solve problems for ArcGIS with Python

**Eric Pimpler**

**[PACKT]**  
PUBLISHING

# **Programming ArcGIS 10.1 with Python Cookbook**

Over 75 recipes to help you automate geoprocessing tasks, create solutions, and solve problems for ArcGIS with Python

**Eric Pimpler**



BIRMINGHAM - MUMBAI

# **Programming ArcGIS 10.1 with Python Cookbook**

Copyright © 2013 Packt Publishing

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author, nor Packt Publishing, and its dealers and distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

First published: February 2013

Production Reference: 1120213

Published by Packt Publishing Ltd.  
Livery Place  
35 Livery Street  
Birmingham B3 2PB, UK.

ISBN 978-1-84969-444-5

[www.packtpub.com](http://www.packtpub.com)

Cover Image by Sujay Gawand ([sujaygawand@gmail.com](mailto:sujaygawand@gmail.com))

# Credits

**Author**

Eric Pimpler

**Reviewers**

Alissa Bickar

Ann Stark

Tripp Corbin, CFM, GISP

**Acquisition Editor**

Andrew Duckworth

**Lead Technical Editor**

Dayan Hyames

**Technical Editor**

Lubna Shaikh

**Project Coordinator**

Amey Sawant

**Proofreaders**

Chris Brown

Sandra Hopper

**Indexer**

Tejal Soni

**Graphics**

Aditi Gajjar

**Production Coordinators**

Manu Joseph

Nitesh Thakur

**Cover Work**

Nitesh Thakur

# About the Author

**Eric Pimpler** is the founder and owner of GeoSpatial Training Services ([geospatialtraining.com](http://geospatialtraining.com)) and has over 20 years of experience in implementing and teaching GIS solutions using ESRI technology. Currently, Eric focuses on ArcGIS scripting with Python, and the development of custom ArcGIS Server web and mobile applications using JavaScript.

Eric has a Bachelor's degree in Geography from Texas A&M University and a Master's of Applied Geography degree with a concentration in GIS from Texas State University.

# About the Reviewers

**Alissa Bickar** is a GIS Analyst and instructor who has a large interest in geospatial technologies and projects. She has developed various courses as an instructor for GeoSpatial Training Services and has been appointed as the ArcGIS Desktop Training Program Manager with GTS. She is responsible for developing and updating course materials for the program, as well as assisting clients with their course and annual subscriptions.

She has extensive experience in the GIS field as a consultant to federal and local governments, environmental engineering firms, and many clients in the Oil and Gas industry. She also has experience as a college professor and has helped develop GIS and Geography courses for higher education.

Alissa has both a Bachelor's and Master's degree in Geography from California University of Pennsylvania.

**Ann Stark**, a GISP since 2005, has been active in the GIS profession for 15 years. She is passionate about GIS and is an active and engaging member of the GIS community in the Pacific Northwest of the United States, coordinating local user groups and serving as the President of the region's GIS professional group. She is an enthusiastic teacher who explains how to effectively use Python with ArcGIS and maintains a blog devoted to the topic at [GISStudio.wordpress.com](http://GISStudio.wordpress.com). She co-owns a GIS consulting business, called Salish Coast Sciences, which provides strategic planning, process automation, and GIS development services.

To unwind from technology, Ann enjoys spending time with her husband and son at their urban farm in the heart of a city where they seek to live sustainably and as self-sufficiently as an urban farm allows.

**Tripp Corbin, CFM, GISP** is the CEO and a Co-founder of eGIS Associates, Inc. He has over 20 years of surveying, mapping, and GIS-related experience. Tripp is recognized as an industry expert with a variety of geospatial software packages including Esri, Autodesk, and Trimble products. He holds multiple certifications including **Microsoft Certified Professional (MCP)**, **Certified Floodplain Manager (CFM)**, **Certified GIS Professional (GISP)**, **Comptia Certified Technical Trainer (CTT+)**, and **Esri Certified Trainer**.

As a GIS Instructor, Tripp has taught students from around the world the power of GIS. He has authored many classes on topics ranging from the beginner level, such as Introduction to GIS, GIS Fundamentals to more advanced topics such as ArcGIS Server Installation, Configurations and Tweaks. Tripp recently helped the University of North Alabama Continuing Studies Center develop an online GIS Analyst Certificate Program.

Tripp believes in giving back to the profession that has given him so much. As a result, he is heavily active in multiple GIS-oriented professional organizations. He is a past President of Georgia, URISA, and was recently the Keynote Speaker for the Georgia Geospatial Conference. Tripp also serves on the URISA International Board of Directors, in addition to being a member of the GISP Application Review committee and an At-Large GITA Southeast Board Member.

Other contributions Tripp has made to the GIS Profession include helping to draft the new Geospatial Technology Competency Model that was adopted by the US Department of Labor, presenting at various conferences and workshops around the US, and providing help to other GIS professionals around the world on various blogs, lists, and forums.

# www.PacktPub.com

## Support files, eBooks, discount offers and more

You might want to visit [www.PacktPub.com](http://www.PacktPub.com) for support files and downloads related to your book.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at [www.PacktPub.com](http://www.PacktPub.com) and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at [service@packtpub.com](mailto:service@packtpub.com) for more details.

At [www.PacktPub.com](http://www.PacktPub.com), you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.



<http://PacktLib.PacktPub.com>

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can access, read and search across Packt's entire library of books.

## Why Subscribe?

- ▶ Fully searchable across every book published by Packt
- ▶ Copy and paste, print and bookmark content
- ▶ On demand and accessible via web browser

## Free Access for Packt account holders

If you have an account with Packt at [www.PacktPub.com](http://www.PacktPub.com), you can use this to access PacktLib today and view nine entirely free books. Simply use your login credentials for immediate access.





# Table of Contents

<b>Preface</b>	<b>1</b>
<b>Chapter 1: Fundamentals of the Python Language for ArcGIS</b>	<b>7</b>
Using IDLE for Python script development	8
Python language fundamentals	11
Summary	28
<b>Chapter 2: Writing Basic Geoprocessing Scripts with ArcPy</b>	<b>29</b>
Introduction	29
Using the ArcGIS Python window	30
Accessing ArcPy with Python	32
Executing tools from a script	35
Using ArcGIS Desktop help	39
Using variables to store data	42
Accessing ArcPy modules with Python	44
<b>Chapter 3: Managing Map Documents and Layers</b>	<b>47</b>
Introduction	47
Referencing the current map document	48
Referencing map documents on a disk	50
Accessing a data frame	51
Getting a list of layers in a map document	52
Restricting the list of layers	54
Changing the map extent	56
Getting a list of tables	58
Adding layers to a map document	59
Inserting layers into a map document	61
Updating layer symbology	64
Updating layer properties	66

<b>Chapter 4: Finding and Fixing Broken Data Links</b>	<b>73</b>
Introduction	73
Finding broken data sources in your map document and layer files	74
Fixing broken data sources with MapDocument. findAndReplaceWorkspacePaths()	76
Fixing broken data sources with MapDocument.replaceWorkspaces()	79
Fixing individual Layer and Table objects with replaceDataSource()	82
Finding all broken data sources in all map documents in a folder	85
<b>Chapter 5: Automating Map Production and Printing</b>	<b>89</b>
Introduction	89
Creating a list of layout elements	90
Assigning a unique name to layout elements	92
Restricting the layout elements returned by ListLayoutElements()	97
Updating layout element properties	98
Getting a list of available printers	101
Printing maps with PrintMap()	102
Exporting a map to a PDF file	103
Exporting a map to an image file	105
Creating a map book with PDFDocumentCreate() and PDFDocumentOpen()	106
<b>Chapter 6: Executing Geoprocessing Tools from Scripts</b>	<b>109</b>
Introduction	109
Finding geoprocessing tools	110
Retrieving a toolbox alias	114
Executing geoprocessing tools from a script	117
Using the output of a tool as an input to another tool	120
Setting environment variables and examining tool messages	122
<b>Chapter 7: Creating Custom Geoprocessing Tools</b>	<b>125</b>
Introduction	125
Creating a custom geoprocessing tool	125
<b>Chapter 8: Querying and Selecting Data</b>	<b>143</b>
Introduction	143
Constructing proper attribute query syntax	144
Creating feature layers and table views	150
Selecting features and rows with the Select Layer by Attribute tool	154
Selecting features with the Select by Location tool	157
Combining a spatial and attribute query with the Select by Location tool	161

---

<b>Chapter 9: Using the ArcPy Data Access Module to Select, Insert, and Update Geographic Data and Tables</b>	<b>165</b>
Introduction	166
Retrieving features from a feature class with a SearchCursor	168
Filtering records with a where clause	170
Improving cursor performance with geometry tokens	172
Inserting rows with InsertCursor	175
Updating rows with an UpdateCursor	180
Deleting rows with an UpdateCursor	184
Inserting and updating rows inside an edit session	186
Reading geometry from a feature class	190
<b>Chapter 10: Listing and Describing GIS Data</b>	<b>193</b>
Introduction	194
Getting a list of feature classes in a workspace	194
Restricting the list of feature classes returned with a wildcard	196
Restricting the list of feature classes returned with a feature type	197
Getting a list of fields in a feature class or table	198
Using the Describe() function to return descriptive information about a feature class	201
Using the Describe() function to return descriptive information about an image	204
Returning workspace information with the Describe() function	206
<b>Chapter 11: Customizing the ArcGIS Interface with Add-Ins</b>	<b>209</b>
Introduction	209
Downloading and installing the Python Add-In wizard	210
Creating a button add-in	213
Installing and testing an add-in	223
Creating a tool add-in	227
<b>Chapter 12: Error Handling and Troubleshooting</b>	<b>233</b>
Introduction	233
Exploring the default Python error message	234
Adding Python exception handling structures (try/except/finally)	235
Retrieving tool messages with GetMessages()	237
Filtering tool messages by severity level	238
Testing for and responding to specific error messages	240
Returning individual messages with GetMessage()	242

<b>Appendix A: Automating Python Scripts</b>	<b>245</b>
Introduction	245
Running Python scripts from the command line	246
Using sys.argv[ ] to capture command-line input	251
Adding Python scripts to batch files	253
Scheduling batch files to run at prescribed times	254
<b>Appendix B: Five Things Every GIS Programmer Should Know</b>	
<b>How to Do with Python</b>	<b>263</b>
Introduction	263
Reading data from a delimited text file	264
Sending e-mails	267
Retrieving files from an FTP server	271
Creating ZIP files	275
Reading XML files	278
<b>Index</b>	<b>281</b>

# Preface

ArcGIS is an industry-standard geographic information system from ESRI.

This book will show you how to use the Python programming language to create geoprocessing scripts, tools, and shortcuts for the ArcGIS Desktop environment.

This book will make you a more effective and efficient GIS professional, by showing you how to use the Python programming language with ArcGIS Desktop to automate geoprocessing tasks, manage map documents and layers, find and fix broken data links, edit data in feature classes and tables, and much more.

*Programming ArcGIS 10.1 with Python Cookbook* starts by covering fundamental Python programming concepts in an ArcGIS Desktop context. Using a how-to instruction style, you'll then learn how to use Python to automate common important ArcGIS geoprocessing tasks.

In this book, you will also cover specific ArcGIS scripting topics that will help save you time and effort when working with ArcGIS. Topics include managing map document files, automating map production and printing, finding and fixing broken data sources, creating custom geoprocessing tools, and working with feature classes and tables, among others.

In *Programming ArcGIS 10.1 with Python Cookbook*, you'll learn how to write geoprocessing scripts using a pragmatic approach designed around accomplishing specific tasks in a cookbook style format.

## What this book covers

*Chapter 1, Fundamentals of the Python Language for ArcGIS*, will cover many of the basic language constructs found in Python. Initially, you'll learn how to create new Python scripts or edit existing scripts. From there, you'll get into language features, such as adding comments to your code, variables, and the built-in typing systems that makes coding with Python easy and compact. Furthermore, we'll look at the various built-in data-types that Python offers, such as strings, numbers, lists, and dictionaries. In addition to this, we'll cover statements, including decision support and looping structures for making decisions in your code and/or looping through a code block multiple times.

*Chapter 2, Writing Basic Geoprocessing Scripts with ArcPy*, will teach the basic concepts of the ArcPy Python site package for ArcGIS, including an overview of the basic modules, functions, and classes. The reader will be able write a geoprocessing script using ArcPy with Python.

*Chapter 3, Managing Map Documents and Layers*, will use the Arcpy Mapping module to manage map document and layer files. You will learn how to add and remove geographic layers from map document files, insert layers into data frames, and move layers around within the map document. The reader will also learn how to update layer properties and symbology.

*Chapter 4, Finding and Fixing Broken Data Links*, will teach how to generate a list of broken data sources in a map document file and apply various Arcpy Mapping functions to fix these data sources. The reader will learn how to automate the process of fixing data sources across many map documents.

*Chapter 5, Automating Map Production and Printing*, will teach how to automate the process of creating production-quality maps. These maps can then be printed, exported to image file formats, or exported to PDF files for inclusion in map books.

*Chapter 6, Executing Geoprocessing Tools from Scripts*, will teach how to write scripts that access and run geoprocessing tools provided by ArcGIS.

*Chapter 7, Creating Custom Geoprocessing Tools*, will teach how to create custom geoprocessing tools that can be added to ArcGIS and shared with other users. Custom geoprocessing tools are attached to a Python script that process or analyze geographic data in some way.

*Chapter 8, Querying and Selecting Data*, will teach how to execute the **Select by Attribute** and **Select by Location** geoprocessing tools from a script to select features and records. The reader will learn how to construct queries that supply an optional `where` clause for the **Select by Attribute** tool. The use of feature layers and table views as temporary datasets will also be covered.

*Chapter 9, Using the ArcPy Data Access Module to Select, Insert, and Update Geographic Data and Tables*, will teach how to create geoprocessing scripts that select, insert, or update data from geographic data layers and tables. With the new ArcGIS 10.1 Data Access module, geoprocessing scripts can create in-memory tables of data, called cursors, from feature classes and tables. The reader will learn how to create various types of cursors including search, insert, and update

*Chapter 10, Listing and Describing GIS Data*, will teach how to obtain descriptive information about geographic datasets through the use of the Arcpy Describe function. As the first step in a multi-step process, geoprocessing scripts frequently require that a list of geographic data be generated followed by various geoprocessing operations that can be run against these datasets.

*Chapter 11, Customizing the ArcGIS Interface with Add-Ins*, will teach how to customize the ArcGIS interface through the creation of Python add-ins. Add-ins provide a way of adding user interface items to ArcGIS Desktop through a modular code base designed to perform specific actions. Interface components can include buttons, tools, toolbars, menus, combo boxes, tool palettes, and application extensions. Add-ins are created using Python scripts and an XML file that define how the user interface should appear.

*Chapter 12, Error Handling and Troubleshooting*, will teach how to gracefully handle errors and exceptions as they occur while a geoprocessing script is executing. Arcpy and Python errors can be trapped with the Python `try/except` structure and handled accordingly.

*Appendix A, Automating Python Scripts*, will teach how to schedule geoprocessing scripts to run at a prescribed time. Many geoprocessing scripts take a long time to fully execute and need to be scheduled to run during non-working hours on a regular basis. The reader will learn how to create batch file containing geoprocessing scripts and execute these at a prescribed time.

*Appendix B, Five Things Every GIS Programmer Should Know How to Do with Python*, will teach how to write scripts that perform various general purpose tasks with Python. Tasks, such as reading and writing delimited text files, sending e-mails, interacting with FTP servers, creating ZIP files, and reading and writing JSON and XML files are common. Every GIS programmer should know how to write Python scripts that incorporate this functionality.

## **What you need for this book**

To complete the exercises in this book, you will need to have installed ArcGIS Desktop 10.1 at either the Basic, Standard, or Advanced license level. Installing ArcGIS Desktop 10.1 will also install Python 2.7 along with the IDLE Python code editor.



## Who this book is for

*Programming ArcGIS 10.1 with Python Cookbook* is written for GIS professionals who wish to revolutionize their ArcGIS workflow with Python. Whether you are new to ArcGIS or a seasoned professional, you almost certainly spend time each day performing various geoprocessing tasks. This book will teach you how to use the Python programming language to automate these geoprocessing tasks and make you a more efficient and effective GIS professional.

## Conventions

In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

Code words in text are shown as follows: "we have loaded the `ListFeatureClasses.py` script with IDLE."

A block of code is set as follows:

```
import arcpy
fc = "c:/ArcpyBook/data/TravisCounty/TravisCounty.shp"

# Fetch each feature from the cursor and examine the extent properties
and spatial reference
for row in arcpy.da.SearchCursor(fc, ["SHAPE@"]):
    # get the extent of the county boundary
    ext = row[0].extent
    # print out the bounding coordinates and spatial reference
    print "XMin: " + ext.XMin
    print "XMax: " + ext.XMax
    print "YMin: " + ext.YMin
    print "YMax: " + ext.YMax
    print "Spatial Reference: " + ext.spatialReference.name
```

When we wish to draw your attention to a particular part of a code block, the relevant lines or items are set in bold:

```
import arcpy

fc = "c:/data/city.gdb/streets"

# For each row print the Object ID field, and use the SHAPE@AREA
# token to access geometry properties

with arcpy.da.SearchCursor(fc, ("OID@", "SHAPE@AREA")) as cursor:
    for row in cursor:
        print("Feature {0} has an area of {1}".format(row[0], row[1]))
```

Any command-line input or output is written as follows:

```
[<map layer u'City of Austin Bldg Permits'>, <map layer u'Hospitals'>,
<map layer u'Schools'>, <map layer u'Streams'>, <map layer u'Streets'>,
<map layer u'Streams_Buff'>, <map layer u'Floodplains'>, <map layer
u'2000 Census Tracts'>, <map layer u'City Limits'>, <map layer u'Travis
County'>]
```

**New terms** and **important words** are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this: "go to **Start | Programs | ArcGIS | Python 2.7 | IDLE**".



Warnings or important notes appear in a box like this.



Tips and tricks appear like this.

## Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply send an e-mail to [feedback@packtpub.com](mailto:feedback@packtpub.com), and mention the book title in the subject of your message.

If there is a book that you need and would like to see us publish, please send us a note in the **SUGGEST A TITLE** form on [www.packtpub.com](http://www.packtpub.com) or e-mail [suggest@packtpub.com](mailto:suggest@packtpub.com).

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide on [www.packtpub.com/authors](http://www.packtpub.com/authors).

## Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

## Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at <http://www.PacktPub.com>. If you purchased this book elsewhere, you can visit <http://www.PacktPub.com/support> and register to have the files e-mailed directly to you.

## Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in the text or the code—we would be grateful if you would report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting <http://www.packtpub.com/support>, selecting your book, clicking on the **errata submission form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded on our website, or added to any list of existing errata, under the Errata section of that title. Any existing errata can be viewed by selecting your title from <http://www.packtpub.com/support>.

## Piracy

Piracy of copyright material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works, in any form, on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at [copyright@packtpub.com](mailto:copyright@packtpub.com) with a link to the suspected pirated material.

We appreciate your help in protecting our authors and our ability to bring you valuable content.

## Questions

You can contact us at [questions@packtpub.com](mailto:questions@packtpub.com) if you are having a problem with any aspect of the book, and we will do our best to address it.

# 1

## Fundamentals of the Python Language for ArcGIS

Python supports many of the programming constructs found in other languages. In this chapter, we'll cover many of the basic language constructs found in Python. Initially, we'll cover how to create new Python scripts and edit existing scripts. From there, we'll delve into language features, such as adding comments to your code, creating and assigning data to variables, and built-in variable typing with Python, which makes coding with Python easy and compact.

Next, we'll look at the various built-in data-types that Python offers, such as strings, numbers, lists, and dictionaries. Classes and objects are a fundamental concept in object-oriented programming and in the Python language. We'll introduce you to these complex data structures, which you'll use extensively when you write geoprocessing scripts with ArcGIS.

In addition, we'll cover statements including decision support and looping structures for making decisions in your code and/or looping through a code block multiple times along with `with` statements, which are used extensively with the new `cursor` objects in the Arcpy Data Access module. Finally, you'll learn how to access modules that provide additional functionality to the Python language. By the end of this chapter, you will have learned the following:

- ▶ How to create and edit new Python scripts
- ▶ Python language features
- ▶ Comments and data variables
- ▶ Built-in datatypes (Strings, Numbers, Lists, and Dictionaries)
- ▶ Complex data structures
- ▶ Looping structures
- ▶ Additional Python functionality

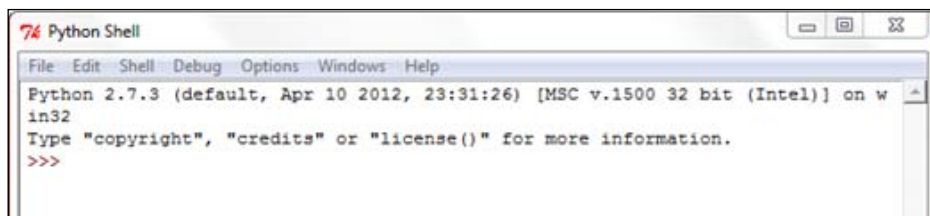
## Using IDLE for Python script development

As I mentioned in the preface, when you install ArcGIS Desktop, Python is also installed along with a tool called IDLE that allows you to write your own code. **IDLE** stands for **I**ntegrated **D**evelopment **E**nvironment. Because it is available with every ArcGIS Desktop installation, we'll use the IDLE development environment for many of the scripts that we write in this book along with the Python window embedded in ArcGIS Desktop. As you progress as a programmer, you may find other development tools that you prefer over IDLE. You can write your code in any of these tools.

### The Python shell window

To start the IDLE development environment for Python, you can go to **Start | Programs | ArcGIS | Python 2.7 | IDLE**. Please note that the version of Python installed with ArcGIS will differ depending upon the ArcGIS version that you have installed. For example, ArcGIS 10.0 uses Python 2.6 while ArcGIS 10.1 uses Python 2.7.

A Python shell window similar to the screenshot will be displayed:

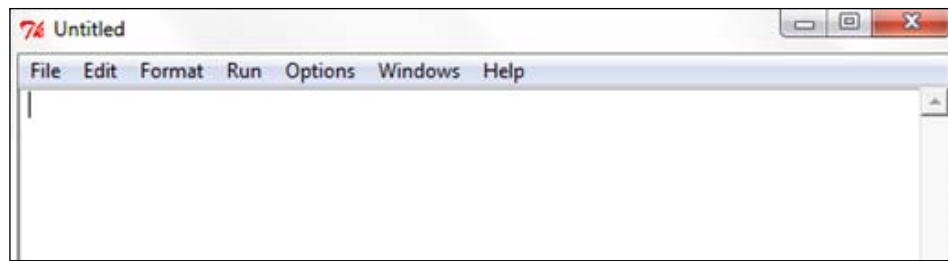


The Python shell window is used for output and error messages generated by scripts. A common mistake for beginners is to assume that the geoprocessing scripts will be written in this shell window. That is not the case. You will need to create a separate code window to hold your scripts.

Although the shell window isn't used to write entire scripts, it can be used to interactively write code and get immediate feedback. ArcGIS has a built-in Python shell window that you can use in much the same way. We'll examine the ArcGIS Python window in the next chapter.

### The Python script window

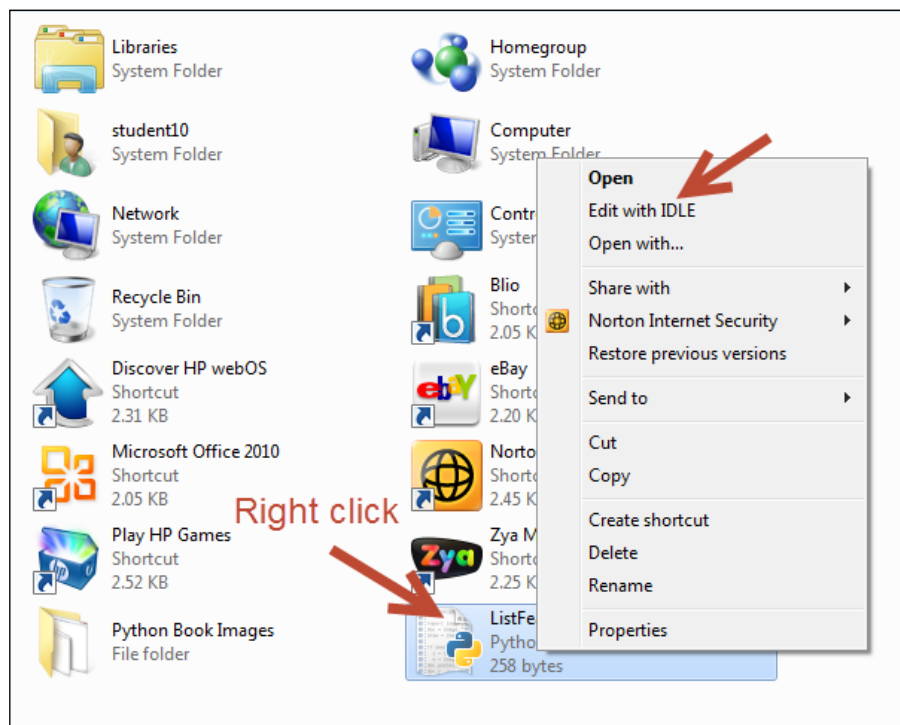
Your scripts will be written in IDLE inside a separate window known as the **Python script window**. To create a new code window, select **File | New Window** from the IDLE shell window. A window similar to that in the following screenshot will be displayed:



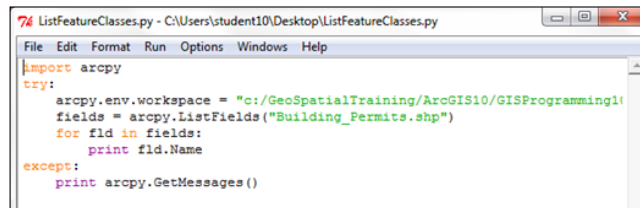
Your Python scripts will be written inside this new code window. Each script will need to be saved to a local or network drive. By default, scripts are saved with a `.py` file extension.

## Editing existing Python scripts

Existing Python script files can be opened from Windows Explorer by right-clicking on the file and selecting **Edit with IDLE**, which brings up a new shell window along with the script loaded in the Python script editor. You can see an example of this in the following screenshot:



In this instance, we have loaded the `ListFeatureClasses.py` script with IDLE. The code is loaded inside the script window:



```
74 ListFeatureClasses.py - C:\Users\student10\Desktop\ListFeatureClasses.py
File Edit Format Run Options Windows Help
import arcpy
try:
    arcpy.env.workspace = "c:/GeoSpatialTraining/ArcGIS10/GISProgramming1/
    fields = arcpy.ListFields("Building_Permits.shp")
    for fld in fields:
        print fld.Name
except:
    print arcpy.GetMessages()
```

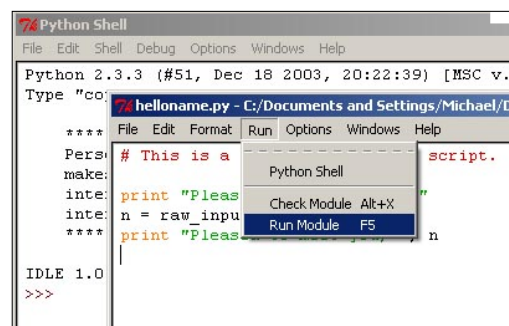
Now that the code window is open, you can begin writing or editing code. You can also perform some basic script debugging with the IDLE interface. Debugging is the process of identifying and fixing errors in your code.

## Executing scripts from IDLE

Once you've written a geoprocessing script in the IDLE code window or opened an existing script, you can execute the code from the interface. IDLE does provide functionality that allows you to check the syntax of your code before running the script. In the code window, select **Run | Check Module** to perform a syntax check of your code.

Any syntax errors will be displayed in the shell window. If there aren't any syntax errors, you should just see the prompt in the shell window. While the IDLE interface can be used to check for syntax errors, it doesn't provide a way of checking for logical errors in your code nor does it provide more advanced debugging tools found in other development environments, such as PythonWin or Wingware.

Once you're satisfied that no syntax errors exist in your code, you can run the script. Select **Run | Run Module** to execute the script:



```
74 Python Shell
File Edit Shell Debug Options Windows Help
Python 2.3.3 (#51, Dec 18 2003, 20:22:39) [MSC v.1310 64-bit (AMD64)]
Type "code" to enter interactive mode.
>>>
**** File Edit Format Run Options Windows Help
# This is a script.
make: Python Shell
inte: print "Please enter a name: "
inte: n = raw_input()
**** print "Please enter a name: "
IDLE 1.0
>>>
```

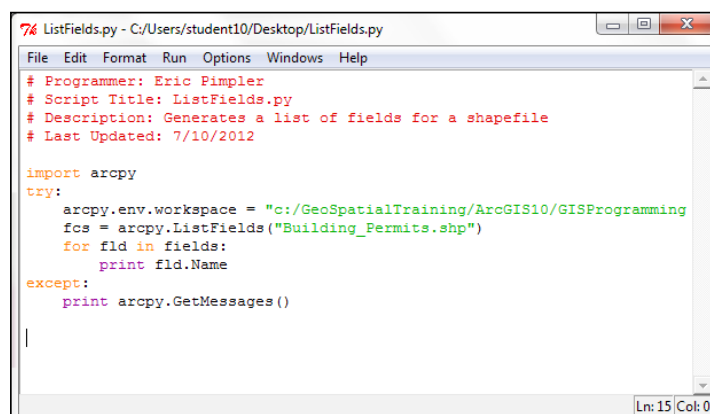
Any error messages will be written to the shell window along with output from `print` statements and system-generated messages. The `print` statement simply outputs a string to the shell window. It is often used for updating the status of a running script or for debugging the code.

## Python language fundamentals

To effectively write geoprocessing scripts for ArcGIS, you are going to need to understand at least the basic constructs of the Python language. Python is easier to learn than most other programming languages, but it does take some time to learn and effectively use it. This section will teach you how to create variables, assign various datatypes to variables, understand the different types of data that can be assigned to variables, use different types of statements, use objects, read and write files, and import third-party Python modules.

### Commenting code

Python scripts should follow a common structure. The beginning of each script should serve as documentation detailing the script name, author, and a general description of the processing provided by the script. This documentation is accomplished in Python through the use of comments. Comments are lines of code that you add to your script that serve as a documentation of what functionality the script provides. These lines of code begin with a single pound sign (#) or a double pound sign (##), and are followed by whatever text you need to document the code. The Python interpreter does not execute these lines of code. They are simply used for documenting your code. In the next screenshot, the commented lines of code are displayed in red. You should also strive to include comments throughout your script to describe important sections of your script. This will be useful to you (or another programmer) when the time comes to update your scripts.



```

74 ListFields.py - C:/Users/student10/Desktop/ListFields.py
File Edit Format Run Options Windows Help
# Programmer: Eric Pimpler
# Script Title: ListFields.py
# Description: Generates a list of fields for a shapefile
# Last Updated: 7/10/2012

import arcpy
try:
    arcpy.env.workspace = "c:/GeoSpatialTraining/ArcGIS10/GISProgramming
    fcs = arcpy.ListFields("Building_Permits.shp")
    for fld in fields:
        print fld.Name
except:
    print arcpy.GetMessages()
|
Ln: 15 Col: 0

```



#### Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at <http://www.PacktPub.com>. If you purchased this book elsewhere, you can visit <http://www.PacktPub.com/support> and register to have the files e-mailed directly to you.



## Importing modules

Although Python includes many built-in functions, you will frequently need to access specific bundles of functionality, which are stored in external modules. For instance, the **Math module** stores specific functions related to processing numeric values and the **R module** provides statistical analysis functions. Modules are imported through the use of the `import` statement. When writing geoprocessing scripts with ArcGIS, you will always need to import the `ArcPy` module, which is the Python package for accessing GIS tools and functions provided by ArcGIS. `import` statements will be the first lines of code (not including comments) in your scripts:

```
import arcpy, os
```

## Variables

At a high level, you can think of a variable as an area in your computer's memory reserved for storing values while the script is running. Variables that you define in Python are given a name and a value. The values assigned to variables can then be accessed by different areas of your script as needed, simply by referring to the variable name. For example, you might create a variable that contains a feature class name, which is then used by the **Buffer** tool to create a new output dataset. To create a variable, simply give it a name followed by the assignment operator, which is just an equal sign (=), and then a value:

```
fcParcels = "Parcels"
fcStreets = "Streets"
```

The following table illustrates the variable name and values assigned to the variable using the preceding code example:

Variable name	Variable value
fcParcels	Parcels
fcStreets	Streets

There are certain naming rules that you must follow when creating variables, including the following:

- ▶ Can contain letters, numbers, and underscores
- ▶ First character must be a letter
- ▶ No special characters in variable name other an underscore
- ▶ Can't use Python keywords

There are a few dozen Python keywords that must be avoided including `class`, `if`, `for`, `while`, and others.

Some examples of legal variable names in Python:

- ▶ `featureClassParcel`
- ▶ `fieldPopulation`
- ▶ `field2`
- ▶ `ssn`
- ▶ `my_name`

Some examples of illegal variable names in Python:

- ▶ `class` (Python keyword)
- ▶ `return` (Python keyword)
- ▶ `$featureClass` (illegal character, must start with a letter)
- ▶ `2fields` (must start with a letter)
- ▶ `parcels&Streets` (illegal character)

Python is a case-sensitive language, so pay particular attention to the capitalization and naming of variables in your scripts. Case-sensitivity issues are probably the most common source of errors for new Python programmers, so always consider this as a possibility when you encounter errors in your code. Let's look at an example. The following is a list of three variables; note that although each variable name is the same, the casing is different, resulting in three distinct variables.

- ▶ `mapsize = "22x34"`
- ▶ `MapSize = "8x11"`
- ▶ `Mapsize = "36x48"`

If you print these variables, you will get the following output:

```
print mapsize
>>> 22x34

print MapSize
>>> 8x11

print Mapsize
>>> 36x48
```

Python variable names need to be consistent throughout the script. Best practice is to use camel casing, wherein the first word of a variable name is all lowercase and then each successive word begins with an uppercase letter. This concept is illustrated in the following example with the variable name `fieldOwnerName`. The first word (`field`) is all lower case followed by an uppercase letter for the second word (`Owner`) and third word (`Name`):

```
fieldOwnerName
```

In Python, variables are dynamically typed. **Dynamic typing** means that you can define a variable and assign data to it without specifically defining that a variable name will contain a specific type of data. Commonly used datatypes that can be assigned to variables include the following:

Datatype	Example value	Code example
String	"Streets"	fcName = "Streets"
Number	3.14	percChange = 3.14
Boolean	True	ftrChanged = true
List	Streets, Parcels, Streams	lstFC = ["Streets", "Parcels", "Streams"]
Dictionary	'0':Streets,'1':Parcels	dictFC = {'0':Streets,'1':Parcels}
Object	Extent	spatialExt = map.extent

We will discuss each of these data-types in greater detail in the coming sections.

For instance, in C# you would need to define a variable's name and type before using it. This is not necessary in Python. To use a variable, simply give it a name and value, and you can begin using it right away. Python does the work behind the scenes to figure out what type of data is being held in the variable.

For example, in C# .NET you would need to name and define the datatype for a variable before working with the variable. In the following code example, we've created a new variable called `aTouchdown`, which is defined as an integer variable, meaning that it can contain only integer data. We then assign the value 6 to the variable:

```
int aTouchdown;  
aTouchdown = 6;
```

In Python, this same variable can be created and assigned data through dynamic typing. The Python interpreter is tasked with dynamically figuring out what type of data is assigned to the variable:

```
aTouchdown = 6
```

Your Python geoprocessing scripts for ArcGIS will often need to reference the location of a dataset on your computer or perhaps a shared server. References to these datasets will often consist of paths stored in a variable. In Python, pathnames are a special case that deserve some extra mention. The backslash character in Python is a reserved escape character and a line continuation character, thus there is a need to define paths using two back slashes, a single forward slash, or a regular single backslash prefixed with the letter `r`. These pathnames are always stored as strings in Python. You'll see an example of this in the following section.

Illegal path reference:

```
fcParcels = "c:\Data\Parcels.shp"
```

Legal path references:

```
fcParcels = "c:/Data/Parcels.shp"
fcParcels = "c:\\Data\\Parcels.shp"
fcParcels = r"c:\Data\Parcels.shp"
```

There may be times when you know that your script will need a variable, but don't necessarily know ahead of time what data will be assigned to the variable. In these cases, you could simply define a variable without assigning data to it. Data that is assigned to the variable can also be changed while the script is running.

Variables can hold many different kinds of data including primitive datatypes such as strings and numbers along with more complex data, such as lists, dictionaries and even objects. We're going to examine the different types of data that can be assigned to a variable along with various functions that are provided by Python for manipulating the data.

## Built-in datatypes

Python has a number of built-in data-types. The first built-in type that we will discuss is the `string` data-type. We've already seen several examples of `string` variables, but these types of variables can be manipulated in a lot of ways, so let's take a closer look at this data-type.

### Strings

Strings are ordered collections of characters used to store and represent text-based information. This is a rather dry way of saying that string variables hold text. String variables are surrounded by single or double quotes when being assigned to a variable. Examples could include a name, feature class name, a `where` clause, or anything else that can be encoded as text.

### String manipulation

Strings can be manipulated in a number of ways in Python. String concatenation is one of the more commonly used functions and is simple to accomplish. The `+` operator is used with string variables on either side of the operator to produce a new string variable that ties the two string variables together:

```
shpStreets = "c:\\GISData\\Streets" + ".shp"
print shpStreets
```

Running this code example produces the following result:

```
>>>c:\GISData\Streets.shp
```

String equality can be tested using Python's `==` operator, which is simply two equal signs placed together. Don't confuse the equality operator with the assignment operator, which is a single equal to sign. The equality operator tests two variables for equality, while the assignment operator assigns a value to a variable:

```
firstName = "Eric"
lastName = "Pimpler"
firstName == lastname
```

Running this code example produces the following result:

```
>>>False
```

Strings can be tested for containment using the `in` operator, which returns `True` if the first operand is contained in the second.

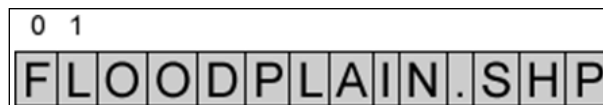
```
fcName = "Floodplain.shp"
print ".shp" in fcName
>>>True
```

I briefly mentioned that strings are an ordered collection of characters. What does this mean? It simply means that we can access individual characters or a series of characters from the string. In Python, this is referred to as **indexing** in the case of accessing an individual character, and **slicing** in the case of accessing a series of characters.

Characters in a string are obtained by providing the numeric offset contained within square brackets after a string. For example, you could obtain the first string character in the `fc` variable by using the syntax `fc[0]`. Negative offsets can be used to search backwards from the end of a string. In this case, the last character in a string is stored at index `-1`. Indexing always creates a new variable to hold the character:

```
fc = "Floodplain.shp"
print fc[0]
>>>'F'
print fc[10]
>>> '.'
print fc[13]
>>> 'p'
```

The following image illustrates how strings are an ordered collection of characters with the first character occupying position **0**, the second character occupying position **1**, and each successive character occupying the next index number:



While string indexing allows you to obtain a single character from a string variable, string slicing enables you to extract a contiguous sequence of strings. The format and syntax is similar to indexing, but with the addition of a second offset, which is used to tell Python how many characters to return.

The following code example provides an example of string slicing. The `theString` variable has been assigned a value of `Floodplain.shp`. To obtain a sliced variable with the contents of `Flood`, you would use the `theString[0:5]` syntax:

```
theString = "Floodplain.shp"
print theString[0:5]
>>>Flood
```



Python slicing returns the characters beginning with the first offset up to, but not including, the second offset. This can be particularly confusing for new Python programmers and is a common source of error. In our example, the returned variable will contain the characters `Flood`. The first character, which occupies position 0, is `F`. The last character returned is index 4, which corresponds to the character `d`. Notice that index number 5 is not included since Python slicing only returns characters up to but not including the second offset.

Either of the offsets can be left off. This in effect creates a wild card. In the case of `theString[1:]`, you are telling Python to return all characters starting from the second character to the end of the string. In the second case, `theString[:-1]`, you are telling Python to start at character zero and return all characters except the last.

Python is an excellent language for manipulating strings and there are many additional functions that you can use to process this type of data. Most of these are beyond the scope of this text, but in general all of the following string manipulation functions are available:

- ▶ String length
- ▶ Casing functions for conversion to upper and lower case
- ▶ Removal of leading and trailing whitespace
- ▶ Finding a character within a string
- ▶ Replacement of text
- ▶ Splitting into a list of words based on a delimiter
- ▶ Formatting

## Numbers

Python also has built-in support for numeric data including `int`, `long`, `float`, and `complex` values. Numbers are assigned to variables in much the same way as strings, with the exception that you do not enclose the value in quotes and obviously it must be a numeric value.

Python supports all the commonly used numeric operators including addition, subtraction, multiplication, division, and modulus or remainder. In addition, functions for returning the absolute value, conversion of strings to numeric datatypes, and rounding are also available.

Although Python provides a few built-in mathematical functions, the `math` module can be used to access a wide variety of more advanced `math` functions. To use these functions, you must specifically import the `math` module as follows:

```
import math
```

Functions provided by the `math` module include those for returning the ceiling and floor of a number, the absolute value, trigonometric functions, logarithmic functions, angular conversion, and hyperbolic functions.

## Lists

A third built-in datatype provided by Python is the `list`. A list is an ordered collection of objects that can hold any type of data supported by Python as well as being able to hold multiple datatypes at the same time. This could be numbers, strings, other lists, dictionaries, or objects. So, for instance, a list variable could hold numeric and string data at the same time. Lists are zero-based, with the first element in the list occupying position 0. Each successive object in the list is incremented by one. In addition, lists have the special capability of dynamically growing and shrinking.

Lists are created by assigning a series of values enclosed by brackets. To pull a value from a list, simply use an integer value in brackets along with the variable name. The following code example provides an illustration of this. You can also use slicing with lists to return multiple values. **Slicing** a list always returns a new list variable.

```
fcList = ["Hydrants", "Water Mains", "Valves", "Wells"]
fc = fcList[0]
print fc
>>>Hydrants
fc = fcList[3]
print fc
>>>Wells
```

Lists are dynamic in nature, enabling them to grow, shrink, and change contents. This is all done without the need to create a new copy of the list. Changing values in a list can be accomplished either through indexing or slicing. Indexing allows you to change a single value, while slicing allows you to change multiple list items.