Chapter 1
Introducing Python

1.1 Introduction

Python is a programming language that is both simple and powerful. For those who have struggled to learn programming languages, this fact may come as a pleasant surprise. This chapter describes some of the main features of Python and its use as a programming language to write scripts for ArcGIS® Pro. The logic and structure of this book is described, followed by some examples of how Python is used.

This book represents a completely updated and revised version of *Python Scripting for ArcGIS*, published by Esri Press (Redlands, California) in 2013, which was written for ArcGIS Desktop 10.*x*. The current book is written for ArcGIS Pro. Not only does it incorporate the changes in the ArcGIS software and the Python programming language, it also includes functionality not previously available in ArcGIS.

1.2 Exploring the features of Python

Python has several features that make it the programming language of choice for working with ArcGIS Pro. Among them:

It is *simple and easy to learn*: Python is easy to learn compared with other highly structured programming languages such as C++. The *syntax* is simple, which gives you more time to focus on solving problems rather than learning the language itself.

It is *free and open source*: Python is *free and open-source software (FOSS)*. You can freely distribute copies of the software, read the source code, make changes to it, and use pieces of it in new free programs. One of the reasons Python works so well is that it has been created, and is constantly being improved, by an active and dedicated user community. The FOSS nature of Python makes it possible for Esri® to distribute Python within its ArcGIS Pro software.

It is *cross-platform*: Python is supported on different platforms, including Windows, macOS, and Linux. Python programs can work on any of these platforms with minimal change or often with no change at all. Because ArcGIS Pro runs only on Windows, it may not seem like a big advantage, but the user community for Python is large, in part because of its cross-platform nature.

It is *interpreted*: Many programming languages require that a program be converted from the source language, such as C++, into binary code that the computer can understand. This conversion requires a compiler with various options. Python is an *interpreted language*, which means it does not need compilation to binary code before you run it. You simply run the program directly from the source code, which makes Python easier to work with and much more portable than some other programming languages.

It is *object oriented*: Python is an *object-oriented programming (OOP)* language. An object-oriented program involves a collection of interacting objects, as opposed to the conventional list of tasks. Many modern programming languages support object-oriented programming.

These features make Python one of the most popular programming languages in the world, and its popularity continues to grow. It has become a widely used language for teaching introductory computer programming courses in colleges and universities. The popularity of Python also means there are numerous resources available to learn it.

Python has many other benefits, including that its open-source nature leads to the development of numerous third-party packages that can be installed to add functionality.

It is important to recognize that Python was not created to work specifically with GIS software or geospatial datasets. It is a general-purpose programming language used for numerous tasks, from developing web apps to creating games and from automating operating system procedures to developing machine learning algorithms. This versatility means that the Python skills you learn in this book will be helpful for other tasks than those related to GIS. And perhaps because of its general-purpose nature, Python is widely used in the geospatial community. This adaptability is, in part, because Python is easy to learn and can be used for many different tasks, but it is also because Python has a large developer community, and a growing library of packages to extend its functionality, including for geospatial tasks. Esri has fully embraced Python as one of the key programming languages to work with ArcGIS Pro. Several other GIS software applications also use Python as a programming language, and your investment in learning Python will therefore pay off beyond its use in ArcGIS Pro.

1.3 Scripting versus programming

Although Python is a programming language, it is often referred to as a "scripting language." So what is the difference? In general, a *scripting language* refers to automating certain functionality within another program, whereas a *programming language* involves the development of more sophisticated multifunctional applications. Scripting is a programming task that allows you to connect diverse existing components to accomplish a new, related task. Scripting is the "glue" that allows you to put various existing elements together. Programming, on the other hand, allows you to build components from scratch, as well as the applications that incorporate these components. Languages that work with these lower-level primitives and the raw resources of the computer are referred to as *system languages*. Examples of system languages include C++ and .NET languages such as C#. Scripting languages use built-in higher-level functions and mask some of the details that a system language manages. Examples of scripting languages include Python, Perl, PHP, R, and Ruby.

Esri, for example, relies primarily on C++ as the programming language to create ArcGIS Pro software and all the different components, or objects, that you find in the software. You can then use .NET languages to write your own software that uses these same objects as well as create your own objects. However, you can also use scripting to access the existing functionality of ArcGIS Pro and connect those functions in new ways to extend that functionality.

One of the strengths of Python is that it is both a scripting language and a programming language, although it does not have quite the depth of a system language such as C++. You can use it for relatively simple scripts as well as more advanced programming tasks. The focus of this book is writing Python *scripts* to carry out tasks in ArcGIS Pro. Python can also be used for application development, but these aspects of using Python are not addressed in this book. Python is used here as an interpreted language to work directly with the existing functionality available in ArcGIS Pro.

1.4 Using scripting in ArcGIS Pro

ArcGIS Pro provides support for the use of Python as a scripting language. Python can access all the tools available in ArcGIS Pro, including those that are part of an extension. This feature makes Python scripting an attractive and efficient method for automating tasks. Although the same automation can be accomplished using a system language such as C++ or a .NET language, scripting often requires much less effort.

Python scripting has become a fundamental tool for GIS professionals to extend the functionality of ArcGIS Pro and automate workflows. In earlier releases of ArcMap™ software, a typical approach was to use the built-in VBA (Visual Basic for Applications) programming tools. With ArcMap 10.*x*, VBA was largely replaced with Python, and the use of Python is only strengthened with ArcGIS Pro. Although application development will continue to employ languages such as those used by .NET, Python has several advantages, especially for GIS professionals who are not full-time programmers.

Python is the scripting language of choice to work with ArcGIS Pro and is included in every ArcGIS Pro installation. Python is also directly embedded in many tools in ArcGIS Pro. For example, Python is one of the standard expression types for field calculations. As another example, several geoprocessing tools in ArcGIS Pro consist of Python scripts, even though the casual user does not necessarily notice this fact (or need to). Esri has officially embraced Python as the preferred scripting tool for working with desktop ArcGIS, and ArcGIS Pro has seen further integration of Python within the desktop software interface.

1.5 Python history and versions

Python was created by Guido van Rossum at the Centrum voor Wiskunde en Informatica (CWI) in the Netherlands and was first released in 1991. Van Rossum remained active in the ongoing development of Python but was joined by numerous contributors. The name Python was inspired by the British comedy group Monty Python—it is a common misconception that the name has anything to do with the snake species.

Python's features include basic *data types* such as strings, numbers, lists, and dictionaries, as well as many advanced elements common in object-oriented programming languages. The robustness of Python reflects the need to include the basic features that all programmers need, as well as more advanced functionality that is common in other, more complex programming languages.

Compared with other languages, Python has gone through a limited number of versions, reflecting a philosophy of incremental change and backward compatibility. Python 2 was introduced in 2000, and the most recent (and final) version is 2.7. With every version, new features were added, and over time some inconsistencies and design flaws crept into the code. Python 3 was released in 2008 as a major overhaul, with the primary goal to clean up the code base and remove redundancy. The most recent version at the time of writing is 3.8.

Some of the changes made in Python 3 are fundamental, which result in breaking with the *backward compatibility* philosophy of Python. As a result, not all code written in Python 3 will work in Python 2. Some of the new functionality added in Python 3 has been added to Python 2, a process known as *backporting*. With careful attention to detail, it is therefore possible to write code that works in both versions.

Both versions of Python are currently in use, which can be confusing. The two versions will continue to coexist for some time, but officially Python 2.7 will not be maintained past 2020. Any existing code will continue to work, but after this date, no further improvements will be made to version 2.7. Chapter 4 covers the fundamentals of the Python language. Important for now is to address how the two versions relate to ArcGIS Pro.

Note: *If you are new to learning Python, you should focus on learning Python 3, which is the focus of this book.*

When Python was first selected as the scripting language to work with ArcMap 9.0, the version selected was Python 2. Since then, every new version of ArcGIS Desktop software has installed with the current version of Python 2. For example, ArcGIS Desktop 10.7.1 uses Python 2.7.16. When ArcGIS Pro was developed, it was designed from the beginning to use Python 3. At the time of writing, the most recent release of ArcGIS Pro is version 2.5, and this version works with Python 3.6.9. The choice for Python 3 was logical because of the impending retirement of Python 2, but it was also facilitated by the availability of other Python packages in version 3 that were not available when ArcGIS Desktop 10.0 was introduced in 2010.

The fact that ArcGIS Desktop 10.*x* uses Python 2.7, whereas ArcGIS Pro uses Python 3.6, has many implications. If you are going to write scripts for both versions, you must learn the differences between the two versions of Python, as well as the functionality of ArcGIS Desktop 10.*x* and ArcGIS Pro. The same applies if you are already an experienced Python programmer and are planning to migrate scripts and tools from ArcGIS Desktop 10.*x* to ArcGIS Pro. Many resources and utilities exist to assist with this conversion.

The purpose of this book is to focus on writing scripts for ArcGIS Pro using Python 3. Some of the most relevant differences with ArcGIS Desktop 10.x and Python 2.7 are mentioned in selected chapters, but all the code in the book is written for ArcGIS Pro and Python 3. Although Python code is not 100 percent backward compatible between versions 3 and 2, it is in principle possible to write Python code that works for both versions. However, because of fundamental differences between ArcGIS Desktop 10.*x* and ArcGIS Pro, many scripts written for one version are unlikely to work in the other. Nonetheless, sometimes the differences are small.

If you are relatively new to learning ArcGIS, the only desktop software you may be familiar with is ArcGIS Pro. Good familiarity with ArcGIS Pro is expected for users of this book, but no experience with Python is required. You can essentially ignore the occasional note that refers to ArcGIS Desktop 10.*x* or Python 2.

Note: *Many GIS users will continue to use both ArcGIS Desktop 10.x and ArcGIS Pro for some time to come. At the time of writing, the most current versions are ArcMap 10.7.1 and ArcGIS Pro 2.5. The installation of ArcMap 10.7.1 includes the installation of Python 2.7.16, and the installation of ArcGIS Pro 2.5 includes the installation of Python 3.6.9. These two versions can run on the same computer. When working with ArcGIS Pro 2.5, you should use only Python 3.6.9. Chapter 2 covers how to ensure you are using the correct version of Python.*

1.6 The structure of this book

Python Scripting for ArcGIS Pro consists of 11 chapters that explain the structure and syntax of Python and illustrate how to write scripts for ArcGIS Pro. Sample code is provided throughout the text.

Chapter 1 introduces Python, explains why Python is used for writing scripts for ArcGIS Pro, and illustrates several example scripts that were developed using Python.

Chapter 2 explains how to use Python editors to write and test your code. Several different editors are used in this book, including IDLE, PyCharm, and Spyder. It also describes the Python window in ArcGIS Pro, which allows you to run one or several lines of Python code directly from within ArcGIS Pro. Code written in the Python window can be saved as a script, and existing code from a script can be loaded into the Python window.

Chapter 3 introduces the ArcGIS geoprocessing framework, including the use of tools and the ModelBuilder application. Experienced ArcGIS Pro users will be familiar with most of the material, but a good review will be beneficial. Knowing what is possible with the existing set of geoprocessing tools will be helpful in writing effective scripts. Similarly, Python scripts and ModelBuilder are often used in combination, so a good knowledge of ModelBuilder is recommended to get the most out of Python scripting.

Chapter 4 explains the fundamentals of the Python language, including the use of data types and data structures, variables, keywords, statements and expressions, methods, functions, and modules; controlling workflow; and best practices for writing scripts. Chapter 4 covers the basic syntax of Python for the novice Python user. Experienced Python users will be familiar with this material.

Chapter 5 describes ArcPy, which makes it possible to perform many of the tasks commonly carried out in ArcGIS Pro using Python. ArcPy includes numerous modules, classes, and functions, which provide an effective way to integrate ArcGIS Pro and Python. Chapter 5 also includes working with data paths, environment settings, and licenses.

Chapter 6 includes techniques for data exploration, including describing data, as well as working with lists to characterize data before using it in other operations.

Chapter 7 describes error-handling techniques for anticipating common errors and making your code more robust. The code debugging environment, which allows you to test your code interactively, is also covered.

Chapter 8 introduces how to manipulate spatial and tabular data, including the use of cursors, searching for data, and working with tables and fields. The lessons in this chapter make it possible to use Python to run SQL queries on spatial data and to write expressions for field calculations.

Chapter 9 describes how to work with the geometric properties of spatial objects, including how to use the properties of existing features and create new features.

Chapter 10 describes how to work with rasters in Python, including the tools of the Spatial Analyst extension. The Spatial Analyst module of ArcPy contains many specialized map algebra operators and other classes for using rasters in spatial analysis.

Chapter 11 describes the ArcPy mapping module for automating mapping tasks. This chapter includes working with projects, maps, and layers, as well as exporting and printing layouts.

If you are an experienced developer and have already written scripts and tools in Python for ArcGIS Desktop 10.*x*, and are looking to upgrade your skills to make the move to ArcGIS Pro, many of the chapters in this book will cover material you already know. You may want to focus on chapters that cover materials that have changed the most relative to the *Python Scripting for ArcGIS* book that was written for ArcGIS Desktop 10.*x*. These chapters include chapter 2 on working with Python editors and chapter 11 on map scripting.

Several chapters are also expanded and improved substantially compared with the previous book. Chapter 4 on Python fundamentals is completely revised to update all the syntax to Python 3, and new sections are added on working with tuples, dictionaries, and sets; print formatting; Boolean logic; working with long lines of code; and file management. Chapter 8 has a new section on Python expressions for field calculations. Chapter 9 is expanded substantially with more examples on how to work with geometry objects. Chapter 10 on rasters has new sections on the expanded functionality of the Image Analyst and Spatial Analyst extensions, as well as working with the Raster Cell Iterator and using raster functions.

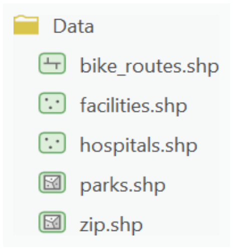
Several important topics with respect to Python scripting for ArcGIS Pro are not included in this book so as to maintain a manageable length. These topics are covered in the book *Advanced Python Scripting for ArcGIS Pro* (Esri Press, 2020). Topics include creating script tools and Python toolboxes; sharing tools with others; managing packages using conda; using third-party packages other than ArcPy; migrating scripts and tools from ArcGIS Desktop 10.*x* to ArcGIS Pro; and using the ArcGIS API for Python and Jupyter Notebook. These topics follow logically on the topics covered in the current book. Therefore, if you have mastered all the topics in *Python Scripting for ArcGIS Pro*, the next logical step is to continue with *Advanced Python Scripting for ArcGIS Pro*.

1.7 Exploring how Python is used

This section uses several examples to illustrate how Python is used to create scripts for use with ArcGIS Pro. These examples resemble the scripts used in later chapters in this book. One of the reasons for presenting these examples is for you to become familiar with looking at Python code. One of the best ways to learn how to write code is to work with existing examples. You are not expected to fully understand the code at this point, but the examples will give you a flavor of what is to come.

Example 1: Copying shapefiles to a geodatabase

A common task in ArcGIS Pro is to copy datasets from one format to another. For example, consider the scenario in which you have several shapefiles in a folder, and you must copy those shapefiles to a new feature dataset inside a file geodatabase. In addition, you want to copy only the polygon shapefiles. The figure shows a typical folder with several shapefiles.



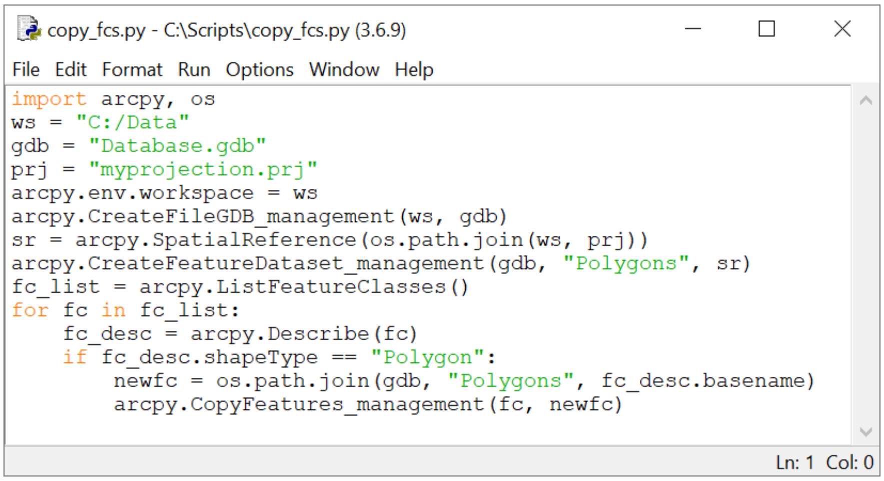
Data
bike_routes.shp
facilities.shp
hospitals.shp
parks.shp
zip.shp

Accomplishing this task in ArcGIS Pro requires running several different tools. You must run Create File Geodatabase and Create Feature Dataset to establish the desired database structure. Then you have several options to copy the shapefiles, including Feature Class to Feature Class, Feature Class to Geodatabase, or Copy Features. In ArcGIS Pro, you typically must run these tools for every shapefile you want to copy. The Feature Class to Geodatabase tool allows you to use multiple shapefiles as input features, but they must be selected manually in the tool dialog box. Running these tools for a handful of shapefiles is not difficult, but what if you had 100 or more shapefiles to copy? Running the tools 100 times would be cumbersome and prone to errors. This is where a Python script will be beneficial.

The following script accomplishes the task. Read through the script to see if you can follow along with the general logic of the script.

```
import arcpy, os
ws = "C:/Data"
gdb = "Database.gdb"
prj = "myprojection.prj"
arcpy.env.workspace = ws
arcpy.CreateFileGDB_management(ws, gdb)
sr = arcpy.SpatialReference(os.path.join(ws, prj))
arcpy.CreateFeatureDataset_management(gdb, "Polygons", sr)
fc_list = arcpy.ListFeatureClasses()
for fc in fc_list:
    fc_desc = arcpy.Describe(fc)
    if fc_desc.shapeType == "Polygon":
        newfc = os.path.join(gdb, "Polygons", fc_desc.basename)
        arcpy.CopyFeatures_management(fc, newfc)
```

The same code is also shown as a completed script in a Python editor.

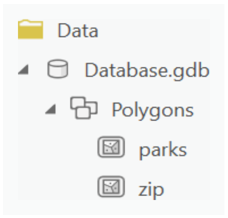

copy_fcs.py - C:\Scripts\copy_fcs.py (3.6.9)
File Edit Format Run Options Window Help
Ln: 1 Col: 0

```
import arcpy, os
ws = "C:/Data"
gdb = "Database.gdb"
prj = "myprojection.prj"
arcpy.env.workspace = ws
arcpy.CreateFileGDB_management(ws, gdb)
sr = arcpy.SpatialReference(os.path.join(ws, prj))
arcpy.CreateFeatureDataset_management(gdb, "Polygons", sr)
fc_list = arcpy.ListFeatureClasses()
for fc in fc_list:
    fc_desc = arcpy.Describe(fc)
    if fc_desc.shapeType == "Polygon":
        newfc = os.path.join(gdb, "Polygons", fc_desc.basename)
        arcpy.CopyFeatures_management(fc, newfc)
```

There is no expectation that you fully understand the code at this point. The next chapters walk you through how to write and execute this type of script. Chapter 2 on Python editors explains where to write this code. Chapter 4 on Python fundamentals explains the syntax used in a Python script. Chapters 5 and 6 cover the fundamentals of ArcPy to use geoprocessing tools in a Python script.

The example script can do more than what geoprocessing tools can do, such as creating a list of all the shapefiles in a folder and filtering out only those that consist of polygons. Without scripting, these are manual tasks in ArcGIS Pro.

The result of running the script is a new geodatabase with a feature dataset containing the copied polygon feature classes.

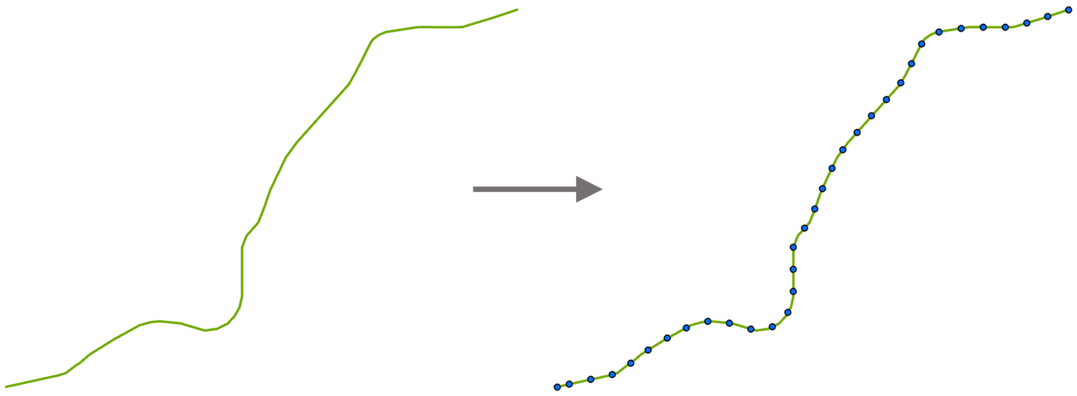


The 14 lines of carefully written code automate the entire task. The script can process any number of inputs because the code is written to read and process the contents of a folder regardless of how many shapefiles are there. The cumbersome and repetitive task of manually copying 100 or more shapefiles is replaced by a single run of the Python script. Of course, it takes skill, time, and effort to develop the script, but once the script is written, it can be used multiple times and revised for different scenarios.

The bottom line is that the time and effort that go into writing a script are typically offset by the time savings that result from using the script to automate repetitive tasks.

Example 2: Creating points along a line

The next example is a bit more complex in terms of the Python code. Consider the example of a feature class containing one or more polylines. The task is to create new points placed along each polyline at regularly spaced intervals. It requires reading the geometry of each feature and creating new points on the basis of the properties of the geometry relative to the interval. The figure shows an example of regularly spaced points along a polyline to illustrate what the Python script accomplishes.



Note: *There is an existing tool in ArcGIS Pro called Generate Points Along Lines that accomplishes this task. This geoprocessing tool is a script tool written in Python. The script used here is a simplified version of this script tool for the purpose of illustration.*

The following script accomplishes the task. Read through the script to see if you can follow along with the general logic. Notice how comments are added to make it easier to understand how the script works. The script includes options to place points by distance or by percentage, and an option to include the endpoints or not.

```
# Import modules and set workspace
import arcpy, os
home = "C:/Data"
arcpy.env.workspace = home
# Script parameters
in_fc = "route.shp"
out_fc = "points9.shp"
interval = 500
use_percent = False
end_points = True
# Create output feature class
desc = arcpy.Describe(in_fc)
sr = desc.spatialReference
arcpy.CreateFeatureclass_management(home, out_fc, "POINT",
                                    "", "", "", sr)
# Add a field to transfer FID from input
fid_name = "NEW_ID"
arcpy.AddField_management(out_fc, fid_name, "LONG")
# Create new points based on input lines
with arcpy.da.SearchCursor(in_fc, ["SHAPE@", "OID@"]) as search_cur:
     with arcpy.da.InsertCursor(out_fc, ["SHAPE@",
                                fid_name]) as insert_cur:
        for row in search_cur:
            line = row[0]
            if line:
                if end_points:
                    insert_cur.insertRow([line.firstPoint, row[1]])
                cur_length = interval
                max_position = 1
                if not use_percent:
                    max_position = line.length
                while cur_length < max_position:
                      insert_cur.insertRow(
                              [line.positionAlongLine(cur_length,
                              use_percent), row[1]])
                    cur_length += interval
                if end_points:
                    insert_cur.insertRow([line.lastPoint, row[1]])
```