1
Introduction to Python for ArcGIS

In this chapter, we will discuss the development of Python as a programming language, from its beginning in the late 1980s to its current state. We will discuss the philosophy of design that spurred its development, and touch on important modules that will be used throughout the book, especially focusing on the modules built into the Python standard library. This overview of the language and its features will help explain what makes Python a great language for ArcGIS automation.

This chapter will cover:

- A quick overview of Python: What it is and does, who created it, and where it is now
- The ArcPy module and other important modules
- Python as a general purpose programming language

Overview of Python

Python, created by *Guido van Rossum* in 1989, was named after his favorite comedy troupe, Monty Python. His work group at the time had a tradition of naming programs after TV shows, and he wanted something irreverent and different from its predecessors - ABC, Pascal, Ada, Eiffel, FORTRAN, and others. So he settled on Python, feeling it was a bit edgy and catchy as well. It's certainly more fun to say than C, the language on which Python is based.

Today, Python is a major programming language. It is used in web development, database administration, and even to program robots. Most importantly to GIS Analysts, Python can be used to control ArcGIS tools and Map Documents to produce geospatial data and maps in an organized and speedy manner using the excellent **ArcPy** module.

ArcPy is installed with ArcGIS for desktop and ArcGIS for server. ArcPy has been the official ArcGIS scripting language since ArcGIS 10.0 and has steadily improved in functionality and implementation. This book will target ArcGIS for Desktop 10.1 and later, and will demonstrate how to make use of Python and its powerful programming libraries (or modules) when crafting complex geospatial analyses.

Python as a programming language

Over the past 40 years, programming languages have developed from assembly and machine code towards high-level abstracted languages that are much closer to English. The Python programming language was designed to overcome many issues that programmers were complaining about in the 1980s: slow development time, overly complicated syntax, and horrible readability. *Van Rossum* wanted to develop a language that could enable rapid code development and testing, have simple or at least readable) syntax, and produce results with fewer lines of code, in less time. The first version of Python (0.9.0) was released in 1991 and was freely obtainable from the start; Python was open source before the term open source was invented.

Interpreted language

Python is an interpreted language. It is written in C, a compiled language, and the code is interpreted from Python into C before it is executed. Practically, this means that the code is executed as soon as it is converted and compiled. While code interpretation can have speed implications for the execution of Python-based programs, the faster development time allowed by Python makes this drawback easy to ignore. Testing of code snippets is much faster in an interpretive environment, and it is perfect to create scripts to automate basic, repeatable computing tasks. Python scripts have the `.py` extentions. Once the code has been interpreted, a second Python script (with the `.pyc` extentions) is generated to save the compiled code. The `.pyc` script will be automatically recompiled when changes are made in the original `.py` script.

Standard (built-in) library

Python, when installed, has a basic set of functionality that is referred to as the standard library. These tools allow Python to perform string manipulations, math computations, and HTTP calls and URL parsing, along with many other functions. Some of the tool libraries, known to Python programmers as modules, are built-in and available as soon as Python is started, while others must be explicitly called using the `import` keyword to make their functions and classes available. Other modules have been developed by third parties and can be downloaded and installed onto the Python installation as needed.

Many new programmers wonder if Python is a real programming language, which is a loaded question. The answer is yes; Python can be used to create complete programs, build websites, run computer networks, and much more. The built-in modules and add-on modules make Python very powerful, and it can be (and has been) used for nearly any part of a computer—operating systems, databases, web servers, desktop applications, and so on. It is not always the best choice for the development of these tools, but that has not stopped programmers from trying and even succeeding.

The glue language

Python is at its best when it is used as a glue language. This term describes the use of Python to control other programs, sending inputs to them and collecting outputs, which are then sent to another program or written to disk. An ArcGIS example would be to use Python to download zipped shapefiles from a website, unzipping the files, processing the files using ArcToolbox, and compiling the results into an Excel spreadsheet. All of this is accomplished using freely available modules that are either included in Python's standard library, or added when ArcGIS is installed.

Wrapper modules

The ArcPy module is a wrapper module. Wrapper modules are common in Python, and are so named because they wrap Python onto the tools we will need. They allow us to use Python to interface with other programs written in C or other programming languages, using the **Application Programming Interface** (**API**) of those programs. For example, wrappers make it possible to extract data from an Excel spreadsheet and transform or load the data into another program, such as ArcGIS. Not all modules are wrappers; some modules are written in pure Python and perform their analysis and computations using the Python syntax. Either way, the end result is that a computer and its programs are available to be manipulated and controlled using Python.

The *Zen of Python* was created to be straightforward, readable, and simplified, compared to other languages that existed previously. This governing philosophy was organized into a poem by *Tim Peters*, an early Python developer called the *Zen of Python*; it is an Easter egg (a hidden feature) included in every Python installation and is shown when `import this` is typed in the Python interpreter:

```
The Zen of Python, by Tim Peters:
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

Go to `https://www.python.org/doc/humor/` for more information.

The basics of Python

Python has a number of language requirements and conventions that allow for the control of modules and structuring of code. The following are a number of important basic concepts, which will be used throughout this book and when crafting scripts for use with geospatial analyses.

Import statements

Import statements are used to augment the power of Python by calling other modules for use in the script. These modules can be part of the standard Python library of modules, such as the math module (used to do higher mathematical calculations) or, importantly, ArcPy, which will allow us to interact with ArcGIS.

`Import` statements can be located anywhere before the module is used, but by convention, they are located at the top of a script.

There are three ways to create an `import` statement. The first, and most standard, is to import the whole module as follows:

```
import arcpy
```

- Using this method, we can even import more than one module on the same line. The following imports three modules: `arcpy`, `os` (the operating system module), and `sys` (the Python system module):

  ```
  import arcpy, os, sys
  ```

- The next method of importing a script is to import a specific portion of a module, instead of importing the entire module, using the `from <module> import <submodule>` syntax:

  ```
  from arcpy import mapping
  ```

- This method is used when only a portion of the code from ArcPy will be needed; it has the practical effect of limiting the amount of memory used by the module when it is called. We can also import multiple portions of the module in the same fashion:

  ```
  from arcpy import mapping, da
  ```

- The third way to import a module is the `from <module> import <submodule>` syntax, but by using an asterisk to import all parts of the module:

  ```
  from arcpy import *
  ```

This last method is still used but it is discouraged as it can have unforeseen consequences. For instance, the names of the variables in the module might conflict with another variable in another module if they are not explicitly imported. For this reason, it is best to avoid this third method. However, lots of existing scripts include import statements of this type so be aware of these consequences.

Variables

Variables are a part of all programming languages. They are used to reference data and store it in memory for use later in a script. There are a lot of arguments over the best method to name variables. No standard has been developed for Python scripting for ArcGIS. The following are some best practices to use when naming variables.

- **Make them descriptive**: Don't just name a variable *x*; that variable will be useless later when the script is reviewed and there is no way to know what it is used for, or why. They should be longer rather than shorter, and should hint at the data they reference or even the data type of the object they reference:

```
shapefilePath = 'C:/Data/shapefile.shp'
```

- **Use camel case to make the variable readable**: Camel case is a term used for variables that start with a lower case letter but have upper case letters in the middle, resembling a camel's hump:

```
camelCase = 'this is a string'
```

- **Include the data type in the variable name**: If the variable contains a string, call it variableString. This is not required, and will not be used dogmatically in this book, but it can help organize the script and is helpful for others who will read these scripts. Python is *dynamically* typed instead of *statically*. A programming language distinction means that a variable does not have to be declared before it can be used, unlike Visual Basic or other statically typed languages. This improves the speed of writing a script, but it can be problematic in long scripts as the data type of a variable will not be obvious.

The ArcGIS does not use camel case when it exports Python scripts, and many examples will not include it; nevertheless, it is recommended when writing new scripts. Also, variables cannot start with a number.

For loops

Built into programming languages is the ability to iterate, or perform a repeating process, over a dataset to transform or extract data that meets specific criteria. Python's main iteration tool is known as a `for` loop. The term `for` loop means that an operation will loop, or iterate, over the items in a dataset to perform the operation on each item. The dataset must be iterable to be used in a `for` loop, a distinction discussed further ahead.

We will be using for loops throughout this book. Here is a simple example that uses the Python Interpreter to take string values and print them in an uppercase format, using a for loop:

```
>>> newlist = [ 'a' , 'b' , 'c' , 'd' ]
>>> for item in newlist:
  print item.upper()
```

The output is shown as follows:

```
A
B
C
D
```

The variable item is a generic variable assigned to each object as it is entered into the for loop, and not a term required by Python. It could have been x or value instead. Within the loop, the first object (`a`) is assigned to the generic variable item and has the upper string function applied to it to produce the output `A`. Once this action has been performed, the next object (`b`) is assigned to the generic variable to produce an output. This loop is repeated for all members of the dataset newlist; once completed, the variable item will still carry the value of the last member of the dataset (`d` in this case).

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.

If/Elif/Else statements

Conditional statements, called if/else statements in Python, are also standard in programming languages. They are used when evaluating data; when certain conditions are met, one action will be taken (the initial if statement; if another condition is met, another action is taken; this is an `elif` statement), and if the data does not meet the condition, a final action is assigned to deal with those cases (the else statement). These are similar to a where conditional in a SQL statement used with the Select tool in ArcToolbox. Here is an example of how to use an if/else statement to evaluate data in a list (a data type discussed further ahead) and find the remainder when divided using the modulus operator (`%`) and Python's is equal to operator (==):

```
>>> data = [1,2,4,5,6,7,10]
>>> for val in data:
    if val % 2 == 0:
    print val,"no remainder"
    elif val % 3 == 2:
    print val, "remainder of two"
    else:
      print "final case"
```

The output is shown as follows:

```
final case
2 no remainder
4 no remainder
5 remainder of two
6 no remainder
final case
10 no remainder
```

While statements

Another important evaluation tool is the `while` statement. It is used to perform an action while a condition is true; when the condition is false, the evaluation will stop. Note that the condition must become false, or the action will be always performed, creating an infinite loop that will not stop until the Python interpreter is shut off externally. Here is an example of using a while loop to perform an action until a true condition becomes false:

```
>>> x = 0
>>> while x < 5:
    print x
    x+=1
```

The output is shown as follows:

```
0
1
2
3
4
```

Comments

Comments in Python are used to add notes within a script. They are marked by a pound sign, and are ignored by the Python interpreter when the script is run. Comments are useful to explain what a code block does when it is executed, or to add helpful notes that script authors would like future script users to read:

```
# This is a comment
```

While it is a programming truism that good code is well-commented code, many programmers skip this valuable step. Also, too many comments can reduce their usefulness and the script's readability. If variables are descriptive enough, and code is well-organized, comments are less necessary; writing the code as verbose and as well-organized as possible will require less time to be spent on comments.

Data types

GIS uses points, lines, polygons, coverages, and rasters to store data. Each of these GIS data types can be used in different ways when performing an analysis and have different attributes and traits. Python, similar to GIS, has data types that organize data. The main data types in Python are strings, integers, floats, lists, tuples, and dictionaries. They each have their own attributes and traits (or properties), and are used for specific parts of code automation. There are also built-in functions that allow for data types to be converted (or casted) from one type to another; for instance, the integer 1 can be converted to the string 1 using the `str()` function:

```
>>> variable = 1
>>> newvar = str(variable)
>>> newvar
```

The output is shown as follows:

```
1
```

Strings

Strings are used to contain any kind of character. They begin and end with quotation marks, with either single or double quotes used, though the string must begin and end with the same type of quotation marks. Within a string, quoted text can appear; it must use the opposite quotation marks to avoid conflicting with the string.Check the following example:

```
>>> quote = 'This string contains a quote: "Here is the quote" '
```

A third type of string is also employed, a multiple line string that starts and ends with three single quote marks:

```
>>> multiString = '''This string has

multiple lines and can go for

as long as I want it too'''
```

Integers

Integers are whole numbers that do not have any decimal places. There is a special consequence to the use of integers in mathematical operations; if integers are used for division, an integer result will be returned. Check out this code snippet below to see an example of this:

```
>>> 5 / 2
```

The output is shown as follows:

```
2
```

Instead of an accurate result of 2.5, Python will return the floor value, or the lowest whole integer for any integer division calculation. This can obviously be problematic and can cause small bugs in scripts that can have major consequences.

Please be aware of this issue when writing scripts and use floats to avoid it as described in the following section.

Floats

Floating point values, or floats, are used by Python to represent decimal values. The use of floats when performing division is recommended:

```
>>> 5.0 / 2
```

The output is shown as follows:

```
2.5
```

Because computers store values in a base 2 binary system, there can be issues representing a floating value that would normally be represented in a base 10 system. Read `docs.python.org/2/tutorial/floatingpoint.html` for a further discussion of the ramifications of this limitation.

Lists

Lists are ordered sets of data that are contained in square brackets ([]). Lists can contain any other type of data, including other lists. Data types can be mixed within a single list. Lists also have a set of methods that allow them to be extended, reversed, sorted, summed, or extract the maximum or minimum value, along with many other methods. Data pieces within a list are separated by commas.

List members are referenced by their index, or position in the list, and the index always starts at zero. Look at the following example to understand this better:

```
>>> alist = ['a','b','c','d']
>>> alist[0]
```

The output is shown as follows:

```
'a'
```

This example shows us how to extract the first value (at the index 0) from the list called `alist`. Once a list has been populated, the data within it is referenced by its index, which is passed to the list in square brackets. To get the second value in a list (the value at index 1), the same method is used:

```
>>> alist[1]
```

The output is shown as follows:

```
'b'
```

To merge two lists, the `extend` method is used:

```
>>> blist = [2,5,6]
>>> alist.extend(blist)
>>> alist
```

The output is shown as follows:

```
['a', 'b', 'c', 'd', 2, 5, 6]
```

Tuples

Tuples are related to lists and are denoted by parentheses (()). Unlike lists, tuples are immutable—they cannot be adjusted or extended once they have been created. Data within a tuple is referenced in the same way as a list, using index references starting at zero:

```
>>> atuple = ('e','d','k')
>>> atuple[0]
```

The output is shown as follows:

```
'e'
```

Dictionaries

Dictionaries are denoted by curly brackets ({}) and are used to create `key:value` pairs. This allows us to map values from a key to a value, so that the value can replace the key and data from the value can be used in processing. Here is a simple example:

```
>>> adic = {'key':'value'}
>>> adic['key']
```

The output is shown as follows:

```
'value'
```

Note that instead of referring to an index position, such as lists or tuples, the values are referenced using a key. Also, keys can be any other type of data except lists (because lists are mutable).

This can be very valuable when reading a shapefile or feature class. Using an `ObjectID` as a key, the value would be a list of row attributes associated with `ObjectID`. Look at the following example to better understand this behavior:

```
>>> objectIDdic = { 1 : [ '100' , 'Main' , 'St' ] }
>>> objectIDdic[1]
```

The output is shown as follows:

```
['100', 'Main', 'St']
```

Dictionaries are very valuable for reading in feature classes and easily parsing through the data by calling only the rows of interest, among other operations. They are great for ordering and reordering data for use later in a script, so be sure to pay attention to them moving forward.

Iterable data types

Lists, tuples, and strings are all iterable data types that can be used in for loops. When entered into a for loop, these data types are operated on in order, unless otherwise specified. For lists and tuples, this is easy to understand, as they have an obvious order:

```
>>> aList = [1,3,5,7]
>>> for value in aList:
    print value * 2
```

The output is shown as follows:

```
2
6
10
14
```

For strings, each character is looped:

```
>>> aString = "esri"
>>> for value in aString:
    print value.upper()
```

The output is shown as follows:

```
E
S
R
I
```

Dictionaries are also iterable, but with a specific implementation that will only allow direct access to the keys of the dictionary (which can then be used to access the values). Also, the keys are not returned in a specific order:

```
>>> aDict = {"key1":"value1",
    "key2":"value2"}
>>> for value in aDict:
    print value, aDict[value]
```

The output is shown as follows:

```
key2 value2
key1 value1
```

Other important concepts

The use of Python for programming requires an introduction to a number of concepts that are either unique to Python but required or common programming concepts that will be invoked repeatedly when creating scripts. Included following are a number of these concepts that must be covered to be fluent in Python.

Indentation

Python, unlike most other programming languages, enforces strict rules on indenting lines of code. This concept is derived again from Guido's desire to produce clean, readable code. When creating functions or using for loops, or if/else statements, indentation is required on the succeeding lines of code. If a for loop is included inside an if/else statement, there will be two levels of indentation. Veteran programmers of other languages have complained about the strict nature of Python's indentation. New programmers generally find it to be helpful as it makes it easy to organize code. Note that a lot of programmers new to Python will create an indentation error at some point, so make sure to pay attention to the indentation levels.

Functions

Functions are used to take code that is repeated over and over within a script, or across scripts, and make formal tools out of them. Using the keyword **def**, short for the define function, functions are created with defined inputs and outputs. The idea of a function in computing is that it takes data in one state and converts it into data in another state, without affecting any other part of the script. This can be very valuable to automate a GIS analysis.

Here is an example of a function that returns the square of any number supplied:

```
def square(inVal):
    return inVal ** 2
>>> square(3)
```

The output is shown as follows:

```
9
```

While this of course duplicates a similar function built into the math module, it shows the basics of a function. A function (generally) accepts data, transforms it as needed, and then returns the new state of the data using the return keyword.

Keywords

There are a number of keywords built into Python that should be avoided when naming variables. These include `max`, `min`, `sum`, `return`, `list`, `tuple`, `def`, `del`, `from`, `not`, `in`, `as`, `if`, `else`, `elif`, `or`, `while`, `and`, `with`, among many others. Using these keywords will result in an error.

Namespaces

Namespaces are a logical way to organize variable names when a variable inside a function (a local variable) shares the same name as a variable outside of the function (a global variable). Local variables contained within a function (either in the script or within an imported module) and global variables can share a name as long as they do not share a namespace.

This issue often arises when a variable within an imported module unexpectedly has the same name of a variable in the script. Python Interpreter will use namespace rules to decide which variable has been called, which can lead to undesirable results.

Zero-based indexing

As mentioned in the preceding section that describes lists and tuples, Python indexing and counting starts at zero, instead of one. This means that the first member of a group of data is at the zero position, and the second member is at the first position, and so on till the last position.

This rule also applies when there is a for loop iteration within a script. When the iteration starts, the first member of the data being iterated is in the zero position.

Also, indexing can be performed when counting from the last member of an iterable object. In this case, the index of the last member is -1, and the second to last is -2, and so on back to the first member of the object.

Important Python Modules for GIS Analysis

Modules, or code libraries that can be called by a script to increase its programming potential, are either built into Python or are created by third parties and added later to Python. Most of these are written in Python, but a number of them are also written in other programming languages and then wrapped in Python to make them available within Python scripts. Modules are also used to make other programs available to Python, such as the tools built in Microsoft Word.

The ArcPy module

The ArcPy module is both a wrapper module used to interact with the ArcGIS tools, which are then executed by ArcGIS in its internal code format, and a code base that allows for additional control of geospatial analyses and map production. ArcPy is used to control the tools in ArcToolbox, but the tools have not been rewritten in Python; instead, we are able to use the ArcGIS tools using ArcPy. ArcPy also gives us the ability to control ArcGIS Map Documents(MXDs) and the objects that MXDs include: legends, titles, images, layers, and the map view itself. ArcPy also has tools that are not available in ArcToolbox. The most powerful of these are the data cursors, especially the new Data Analysis Cursors that create a more Pythonic interface with GIS data. The data cursors, covered extensively in *Chapters 5, ArcPy Cursors: Search, Insert and Update* and *Chapter 6, Working with ArcPy Geometry Objects* are very useful to extract rows of data from data sources for analysis.

The ability to control geospatial analyses using ArcPy allows for the integration of ArcGIS tools into workflows that contain other powerful Python modules. Python's glue language abilities increase the usefulness of ArcGIS by reducing the need to treat geospatial data in a special manner.

The Operating System (OS) module

The OS module, part of the standard library, allows Python to access operating system functionality. A common use of the module is to use the `os.path` method to control file paths by dividing them into directory paths (that is, folders) and base paths (that is, files). There is also a useful method, `os.walk`, which will walk-through a directory and return all files within the folders and subfolders. The OS module is accessed constantly when performing GIS analysis.

The Python System (SYS) module

The sys module, part of the standard library, refers to the Python installation itself. It has a number of methods that will get information about the version of Python installed, as well as information about the script and any arguments (or parameters) supplied to the script, using the `sys.argv` method. The `sys.path` method is very useful to append the Python file path; practically, this means that folders containing scripts can be referenced by other scripts to make the functions they contain importable to other scripts.

The XLRD and XLWT modules

The `XLRD` and `XLWT` modules are used to read and write Excel spreadsheets, respectively. The modules can be very useful to extract data from legacy spreadsheets and convert them into usable data for GIS analysis, or to write analysis results when a geospatial analysis is completed. They are not part of the Python standard library, but are installed along with ArcGIS 10.2 and Python 2.7.

Commonly used built-in functions

There are a number of built-in functions that we will use throughout the book. The main ones are listed as follows:

- `str`: The string function is used to convert any other type of data into a string
- `int`: The integer function is used to convert a string or float into an integer. To not create an error, any string passed to the integer function must be a number such as 1.
- `float`: The float function is used to convert a string or an integer into a float, much like the integer function.

Commonly used standard library modules

The following standard library modules must be imported:

- `datetime`: The datetime module is used to get information about the date and time, and convert string dates into Python dates.
- `math`: The math module is used for higher level math functions that are necessary at times, such as getting a value for Pi or calculating the square of a number.
- `string`: The string module is used for string manipulations.
- `csv`: The CSV module is used to create and edit comma-separated value type files.

Check out `https://docs.python.org/2/library` for a complete list of the built-in modules in the standard library.

Summary

In this chapter, we discussed about the Zen of Python and covered the basics of programming using Python. We began our exploration of ArcPy and how it can be integrated with other Python modules to produce complete workflows. We also discussed the Python standard library and the basic data types of Python.

Next, we will discuss how to configure Python for use with ArcGIS, and explore how to use Integrated Development Environments (IDEs) to write scripts.