Part 1: Learning the fundamentals of Python and geoprocessing

Notes

Contents

Chapter 1: Introducing Python

1.1 Introduction

- This chapter describes the main features of Python

1.2 Exploring the features of Python

- Benefits of Learning Python:
  - It's simple and easy to learn
  - It's free and open source
  - It's cross platform
  - It's interpreted
  - It's object oriented
- A scripting language refers to automating certain functionality with another program
  - Scripting is a programming task that allows you to connect diverse existing components to accomplish a new, related task
- A programming language involves the development of more sophisticated multifunctional applications
- Python is both a scripting and a programming language
  - However, not as detailed as other programs, such as C++
- Python can be used for application development

1.4 Using scripting in ArcGIS

- Python scripting has become a fundamental tool for geographic information systems, particularly for professionals and now within ArcGIS
- Python scripting extends the functionality of ArcGIS and automates workflow
- For example, the Spatial Statistics toolbox is made up of almost all Python scripts

1.5 Python history and versions

- Created by Guido von Rossum at the Centrum voor Wiskunde en Informatica (CWI) in the Netherlands
- First released in 1991
- Features: lists, dictionaries, strings, metaclasses, generators, list comprehensions

1.6 About this book

This book has…

1. The printed book, aka the theory of using Python
2. Digital exercises (disk in the back of the book)

1.7 Exploring how Python is used

Example 1: Determining address errors

- **AddressErrors** script (Bruce Harold) inspects street centerlines for possible errors in address ranges associated with street segments
- Polyline feature class is the output and includes one feature for every error detected. Also contains attribution that profiles the error

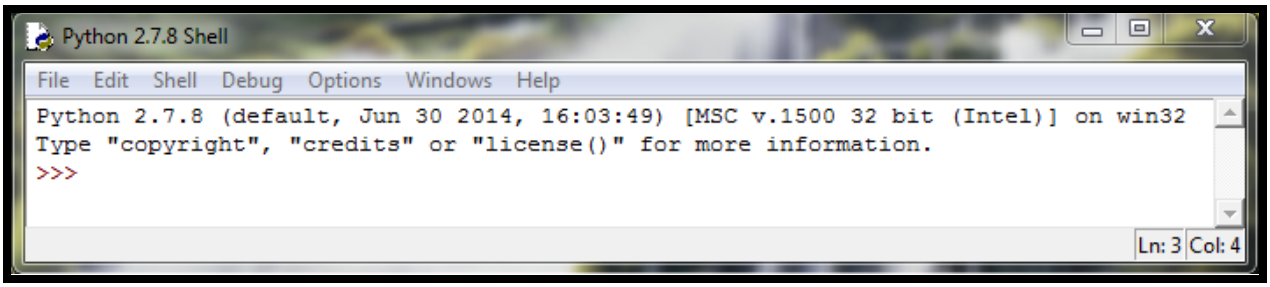
Example 2: Market analysis using the Huff Model Tool

- **Huff Model** script (Drew Flater)
- Written entirely in Python (along with Example 1)
- This is much more complicated (about 700 lines of code)
- Basic elements are the same as less sophisticated scripts

1.8 Choosing a Python script editor

- You can work with Python in a variety of ways
    1. Command line
        a. In Windows, click the Start button, then click All Programs > ArcGIS > Python 2.7 > Python (command line)
        b. Limited support for writing and testing scripts
    2. Integrated Development Environments (IDEs) aka Python editors
        a. The default IDE that comes with an installation of Python is the integrated development environment (IDLE)

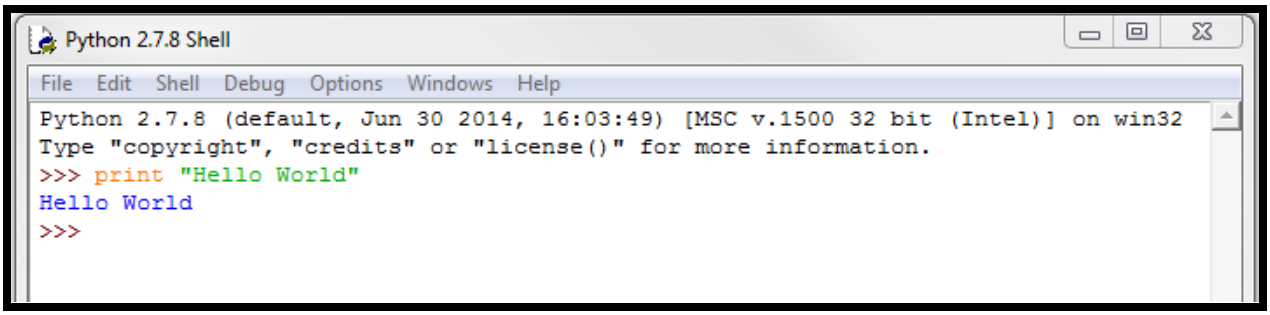
b. To access, click the Start button, click All Programs > ArcGIS > Python 2.7 > IDLE (Python GUI)
c. GUI stands for graphic user interface
d. IDLE is also known as the Python shell
e. For some other Python editors, visit PythonEditors.



- The last line in the Python Shell starts with >>>, which is the prompt of the interactive Python interpreter
  - This is where you can type code and press ENTER
  - This will carry out your command

Let's try an example:

```
>>> print "Hello World"
```



- When you press ENTER, the interactive Python interpreter reads the input command, then prints the string `Hello World` to the next line
- It also gives you a new prompt on the following line
- Printing refers to putting something to the screen
- Syntax highlighting helps by being a way of error checking as you write code
  - Highlighting can vary with Python editors
- If you open a new window in IDLE, and print something, nothing will happen
  - You need to run the file
  - BUT, prior to running the file, you must SAVE it
  - File > Save > example.py
  - Run > Run Module

- It's useful to have the interactive Python interpreter open and you script(s) open at the same time
- A common Python editor in Windows is PythonWin

Chapter 2: Geoprocessing in ArcGIS

2.1 Introduction

- This chapter talks about ArcGIS framework including:
    - ArcToolbox
    - ModelBuilder
    - Python

2.2 What is geoprocessing?

- Geoprocessing in ArcGIS allows you to perform spatial analysis and modeling as well as automate GIS tasks
- A geoprocessing tool takes input data (a feature class, raster, or table), performs a geoprocessing task, and produces output data as a result
- Creating automated workflows combining geoprocessing tools can be accomplished in ArcGIS through the use of models and scripts
- Geoprocessing Framework:
    - A collection of tools, organized in toolboxes and toolsets
    - Methods to find and execute tools, including the Search window, the Catalog window, and the ArcToolbox window
    - Tool dialog boxes for specifying tool parameters in executing tools
    - ModelBuilder for creating models that allow for the sequencing of tools
    - A Python window for executing tools using Python
    - A Results window that logs the geoprocessing tools being executed
    - Methods for creating Python scripts and using them as tools
- Some characteristics:
    - All tools can be accessed from their toolbox, which makes for a consistent procedure for accessing tools, models, and scripts
    - All tools are documented the same way, which allows for consistent cataloging and searching
    - All tools have a similar user interface (the dialog box) for specifying the tools parameters
    - Tools can be shared

2.3 A note on ArcObjects

These aren't covered in this book!

2.4 Using toolboxes and tools

- Geoprocessing tools perform operations on datasets
- Several hundred tools are available in ArcGIS
    - The ones available to you depends on which license you have
- In ArcToolbox, geoprocessing tools are organized into toolboxes, where each box typically contains one or more toolsets, and each toolset contains one or more tools
- Common geoprocessing tools are displayed in the Geoprocessing menu
- You can also search for tools
- You can also take a look in the Catalog menu

2.5 Learning types and categories of tools

There are 4 main types of tools in ArcGIS:

1. Built-in Tools - They are built into ArcObjects and a compiled programming language (aka C++). These are created by Esri (these are the hammer icons)
2. Model Tools - Created using model builder (these are the work-flow chart diagram)
3. Script Tools - Accessible using a tool interface. Carries out geoprocessing operations using, for example, Python scripts (.py) (note paper diagram)
4. Specialized Tools - These are created by system developers

- There are two different categories for tools:
    - System Tools: Created and installed by Esri as part of the regular ArcGIS software
    - Custom Tools: Consist of script and model and are created by the user

2.6 Running tools using tool dialog boxes

I believe that I understand all relevant concepts of this section.

2.7 Specifying environment settings

- Geoprocessing operations are influenced by environment settings, which are like hidden, additional parameters that affect how a tool is run
- The Environment Settings dialog box, go to: Geoprocessing > Environments
- The workspace setting is very important. There are two types of workspaces:
    - The current workspace (where inputs are taken from and outputs are placed)
    - The scratch workspace (used by model tools to write intermediate data)
- Environment settings has a hierarchy of levels:
    - Application
    - Individual tool
    - Model
    - Script
- You can override this hierarchy, if need be

2.8 Using batch processing

- All geoprocessing tools can be run in batch mode
    - Right click on a tool > batch
    - This process means executing a single tool multiple times using different inputs without further intervention
- Batch window shows a grid of rows and columns
    - Columns are parameters of the tool
    - Rows are specified for each run

2.9 Using models and ModelBuilder

- Sometimes you have to run numerous tools to get the result that you are looking for
- Options:
    - Run through each tool, one at a time (has limitations & is repetitive)
    - Use ModelBuilder
- ModelBuilder is a way to create a sequence of tools
    - The output of one tools is the input of another tool
    - Similar to a visual programming language
- Geoprocessing tools are the basic building blocks of a model
- Tools perform geoprocessing operations on geographic data
- Data variables reference data on disk or a layer in the ArcMap table of contents
- Value variables are items such as strings, numbers, Boolean values, spatial references, linear units, and extents
- Connectors are the data and values which are connected to tools. There are 4 types:
    - Data connectors

- Environment connectors
- Precondition connectors
- Feedback connectors

The steps to creating and running model tools in ModelBuilder is as follows:

1. Create a new model
    a. Click on the ModelBuilder Button and make a new model
2. Add data and tools to the model
    a. Drag and drop data
3. Create connectors and fill tool parameters
    a. Connecting parts together
4. Save the model
5. Run the model
6. Examine the model results

2.10 Using scripting

- ModelBuilder and using a text-based scripting language such as Python are very similar in concept
- You can interchange the use of ModelBuilder and Python for certain purposes
- Things you can do in Python but not in ModelBuilder:
    - Lower-level geoprocessing tasks
    - Allows for advanced programming logic
    - Used to wrap other software
    - Run as a stand-alone script
    - Can be run at specific times without user intervention

2.11 Running scripts as tools

- To view the script "under the hood" of script tools, simply right-click the tool and click Edit.
- To run a tool in Python, type the tool name followed by its parameters. For example:

```
>>> import arcpy
>>> arcpy.Clip_analysis("C:/Data/roads.shp", "C:/Data/zipcodes.shp",
"C:/Data/roads_clip.shp")
```

The resulting shapefile is added to the ArcMap table of contents:

```
<Result 'C:\\Data\\roads_clip.shp'>
```

2.12 Converting a model to a script

- To convert models to Python scripts, you must:
- In the ModelBuilder menu bar:
    - click Model > Export > To Python Script
- You cannot do the reverse of this process

2.13 Scheduling a Python script to run at prescribed times

- On Windows 7:
    - Click the Start button, click on Control Panel > Administrative Tools > Task Scheduler
    - OR click on System and Security > Administrative Tools > Task Scheduler
- Double-click Add Scheduled Task (or Create Basic Task)
- Complete the options on the wizard
- Click the Browse button to select the appropriate Python script
- To select arguments for the script to run, click on "Open advanced properties"

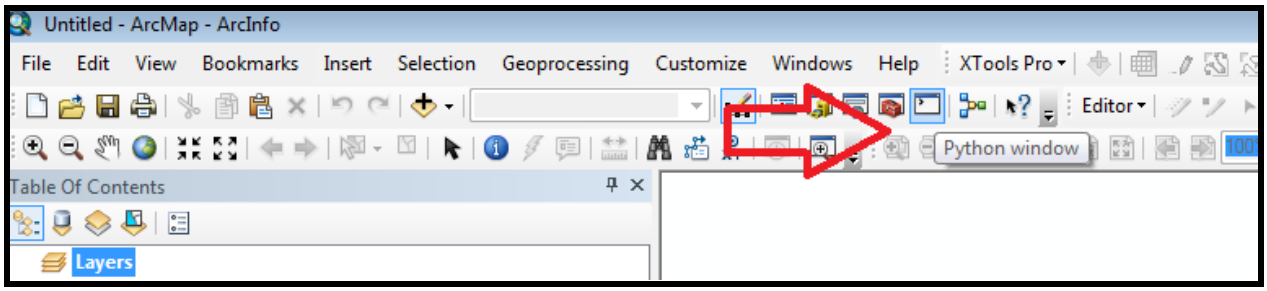
Chapter 3: Using the Python Window

3.1 Introduction

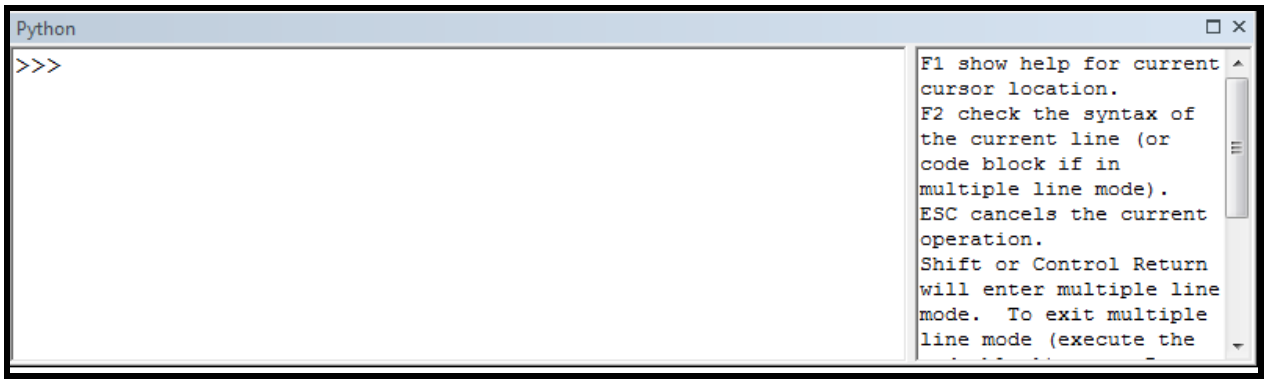
- We're going to learn how to use the Python window in ArcGIS!

3.2 Opening the Python Window

- The Python environment is placed within the "Python Window" in ArcGIS
- In this window, you can run geoprocessing tools while using other Python modules and libraries
- The Python window can be used to run one or more lines of Python code
- This place is good to test syntax and work of short lengths of code
- Scripting ideas can be tested outside a larger script directly within an ArcGIS for Desktop application
- The Python Window can be accessed here:



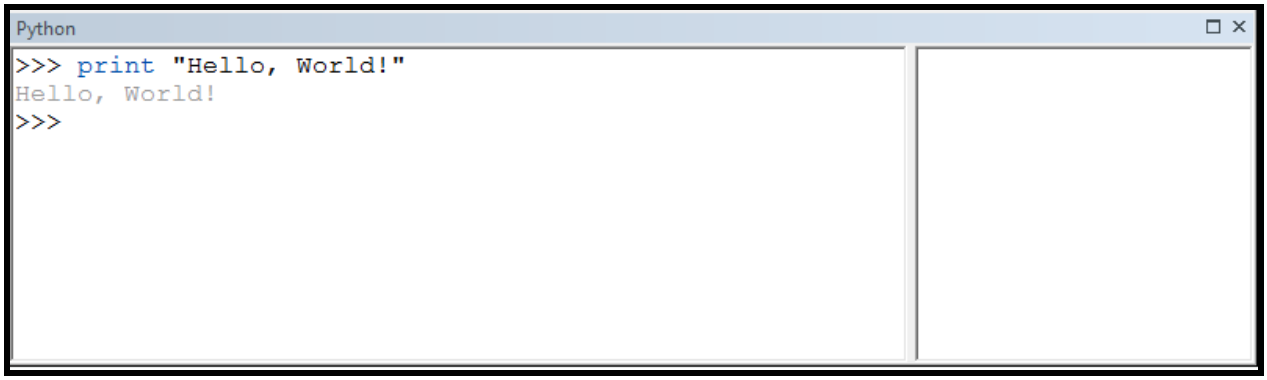
- The left side is the interactive Python interpreter (aka where the Python code is entered)
- The primary prompt is indicated by >>>



- On the right-hand side is the Help and syntax panel, which is updated as code is written
- You can move the box to your liking by standard window editing methods

3.3 Writing and running code

- Python code in the Python window is run one line at a time and the result is immediately displayed



- To force a secondary prompt (if applicable) you simply use the CTRL + ENTER command
- IMPORTANT: All geoprocessing tools can be accessed by importing the **ArcPy** site package. Other non-tool functions such as listing and describing data, working with environment settings, and accessing geoprocessing messages are also available with this package
  - `import arcpy`
- There is an autocompletion option
- Everything you can do in a normal Python shell you can do in the Python window
- Python blocks of code can be written in the Python window and saved in a Python or text file for use in a Python editor, and existing code can be uploaded in the Python window

3.4 Getting Assistance

- For help while in the Python window…
- The F1 key shows Help for the current pointer location
- The F2 key checks the syntax of the current line of code, or block of code for multiline constructs
- The UP ARROW and DOWN ARROW keys can access previously entered commands
- Pressing the TAB key autocompletes an already-established word in the Python document

3.5 Exploring the Python window options

- By right-clicking the Python window, you can…
- Cut, Copy, Paste and Clear provide basic editing lines of code
- Select All allows you to select then copy all lines of code in the Python window
- Clear All allows you to remove all lines of code and start again with an empty Python window
- Show Default Choices autocompletion...or no. Your choice!
- Add to Results includes tools in the Results window that were run in Python
- Load simply loads existing code
- Save As saves code wherever you want
- Help Placement gives you the option on where to place the Help section
- Format provides formatting options

3.6 Saving your work

- You can save files that you make in the Python window in the following formats:
    1. Text file
    2. Python file

3.7 Loading code into the Python window

- You can load pre-existing Python scripts into the Python window

Chapter 4: Learning Python language fundamentals

4.1 Introduction

- This chapter covers the fundamentals of the Python language

4.2 Locating Python documentation and resources

- There's a bunch of resources if I get stuck...check them out sometime (pp. 59 - 60)

4.3 Working with data types and structures

- There are a number different data types:
  - Strings
  - Numbers
  - Lists
  - Tuples
  - Dictionaries
  - More!
- Python also uses different data structures:
  - A collection of data elements that are structured in some way
  - The most basic Python data structure is a sequence - each element is a number or string
- Strings, lists, and numbers are immutable - you can't modify them but only replace them with new values
- Lists and dictionaries are mutable - the data elements can be modified

4.4 Working with numbers

- Numbers can be integers or floats
- Remember that int / int = int and that int / float = float
- This exact solution (with decimal) is called true division

4.5 Working with variables and naming

- Python scripts uses *variables* to store information
- To assign a variable, it would look something like:

```
>>> x = 17
```

- This (above) is called an assignment statement
- Now that the variable has been assigned, you can do work with it:

```
>>> x = 17
>>> x * 2
34
```

- Since the variable that you declare can essentially be any data type, this is known as having a dynamic assignment where you can change the data type later on in the code
- Rules for naming variables:
    - Variable names consist of letters, digits, and underscores (_)
    - Variable names cannot begin with a digit
    - Python keywords cannot be used as variable names
    - Use descriptive names
    - Follow conventions
        - http://www.python.org/dev/peps/pep-0008/
    - Keep it short
    - You should keep a single space around operators
- Multiple variables can be assigned on the same line, such as:

```
>>> x, y, z = 1, 2, 3
>>> print "Is the same as"
>>> x = 1
>>> y = 2
>>> z = 3
```

4.6 Writing statements and expressions

- An expression is a value
- A statement is an instruction that tells the computer to do something

4.7 Using strings

- Converting the value of the variable from one variable to another is known as casting
- Everything else is fairly straight-forward

4.8 Using Lists

- A Python list is surrounded by square brackets [ ]
- Items in the list are separated by commas [,]

4.9 Working with Python objects

- This is also straight-forward

4.10 Using functions

- To print a list of built-in functions, execute the following:

```
>>> print dir(__builtins__)
```

4.11 Using Methods

- Methods are similar to functions
- A method is a function that is closely coupled to an object

```
<object>.<method>(<arguments>)
```

```
>>> topic = "Geographic Information Systems"
>>> topic.count("i")
```

```
2
```

4.12 Working with strings

- I think that, overall, I am good with this section

4.13 Working with Lists

- Ibid

4.14 Working with paths

- In Python, there are three ways to specify a path:
    1. Use a forward slash (/) - "C:\\EsriPress\\Python\\Data"
    2. Use two backslashes (\\) - "C:\\EsriPress\\Python\\Data"
    3. Use a string literal by placing the letter r before a string - for example, r"C:\EsriPress\Python\Data"
        a. The letter r stands for "raw string" which means that the backslash will not be read as an escape character

4.15 Working with modules

- There are many functions available in Python
- To access them, we need to implement modules
- Modules are like extensions that can be imported into Python to extend its capabilities
- Example:

```
>>> import math
```

- To get a list of the functions in the math module (or any), use the dir statement:

```
>>> dir(math)
```

- Make sure to reference modules if necessary!
- Modules are fairly-straightforward but sometimes it can be confusing using potentially two different calling systems

4.16 Controlling workflow using conditional statements

- Branching is one way to control the workflow in your script
  - Means making a decision to take one path or another
  - Typically uses `if` statements

```
import random
x = random.randint(0, 6)
print x
if x == 6:
    print "You win!"
```

- The `if` statements and its variants are called branching structures

4.17 Controlling workflow using loop structures

- For loops and While loops are the main types
- Overall, I feel good about this brief section

4.18 Getting user input

- If the Python script requires inputs from outside the script itself:
  - Use a system argument
    - `sys.argv`
    - Let's the user input something
  - Use the input function
    - `x = input("")`

4.19 Commenting scripts

- There is nothing new here

4.20 Working with code in the PythonWin editor

- This just gives a very brief style guide to PythonWin...I'm not completely sure if I'm going to use it or not

4.21 Following coding guidelines

- This section just reiterates the important stylings that Python uses
- This text frowns against using camelCase rather than underscore_case to name variables...I like it better so I'm going to do it anyway