Python Scripting for ArcGIS

Paul Zandbergen
Department of Geography
University of New Mexico
Outline of Topics

• Introduction
  – Examples, Python and ArcGIS, Python versions
• Fundamentals of geoprocessing in ArcGIS
• Python language fundamentals
  – Where to run Python code
  – Data types: numbers, strings, lists
  – Functions and modules
  – Controlling workflow
• ArcPy: Geoprocessing using Python
  – Using tools, functions, classes
  – Describing data, listing data, working with lists
• Creating custom tools
  – Script tools, tool parameters
• Resources
Workshop Materials Posted

posted until October 24

http://www.paulzandbergen.com/workshops
Forthcoming Book

• Python Scripting for ArcGIS
• Esri Press
• Sometime in 2012
• Updated for ArcGIS 10.1

• Sample exercises posted (for 10.0)
Introduction
Prior Knowledge and Experience

• Using ArcGIS 9.3 or 10.0?
  – Workshop is for 10.0

• Prior Python experience?
  – I’m not assuming any

• Other programming experience?
  – I’m not assuming any
Example 1

• Script to copy all shapefiles in a folder into a geodatabase

```python
import arcpy
from arcpy import env
env.overwriteOutput = True
env.workspace = "c:/workshop/ex01"
fclist = arcpy.ListFeatureClasses()
for fc in fclist:
    fcdesc = arcpy.Describe(fc)
    arcpy.CopyFeatures_management(fc, "c:/workshop/ex01/study.mdb/"
                                + fcdesc.basename)
```
Example 2

- Script tool to generate a k-nearest neighbor table
- Runs an existing ArcGIS tool multiple times, writes the result

```python
import arcpy
from arcpy import env
env.overwriteoutput = True
infc = arcpy.GetParameterAsText(0)
output = arcpy.GetParameterAsText(1)
k = arcpy.GetParameter(2)
n = 1
f = open(output, "w")
while n <= k:
    result = arcpy.CalculateDistanceBand_stats(infc, n)
    f.write(str(n) + " " + str(result[1])+ "\n")
    n = n + 1
f.close()
```
Example 3

- Script tool to run Huff model
- Sophisticated analysis not available in ArcGIS
Example 3

```python
# HuffModel.py
# Created: 4/13/2007 by Drew Flater
# Usage: Creating probability-based trade areas for retail stores
# -----------------------------------------------

# Import system modules
import sys, string, arcgiscripting, os, traceback, shutil, re

# Create the Geoprocessor object
gp = arcgiscripting.create(93)

# Set overwrite
gp.overwriteoutput = 1

def AddPrintMessage(msg, severity):
    print msg
    if severity == 0: gp.AddMessage(msg)
    elif severity == 1: gp.AddWarning(msg)
    elif severity == 2: gp.AddError(msg)

# Start traceback Try-Except statement:
try:
    # Script parameters...
    stores = gp.getparameterastext(0)
    store_name = gp.getparameterastext(1)
    store_attr = gp.getparameterastext(2)
    outputfolder = gp.getparameterastext(3)
    fc_name = gp.getparameterastext(4)
    studyarea = gp.getparameterastext(5)
    blockgroups = gp.getparameterastext(6)
```
What is Python Scripting?

- Add functionality to ArcGIS
  - Integrated into ArcGIS interface
  - Builds upon existing functionality
  - Automates repetitive tasks
  - Expands analysis options

- Share new functionality
  - Script tools work just like regular tools
  - Can be integrated into models, tools
  - Easy to share with others (free)
Why Python?

• Free, open source
• Object oriented
• Basic scripting AND complex object-oriented programming
• “Batteries included”
• Embraced by geospatial community, including ESRI
• Many libraries
Python Programming Language — Official Website

Python is a programming language that lets you work more quickly and integrate your systems more effectively. You can learn to use Python and see almost immediate gains in productivity and lower maintenance costs.

Python runs on Windows, Linux/Unix, Mac OS X, and has been ported to the Java and .NET virtual machines.

Python is free to use, even for commercial products, because of its OSI-approved open source license.

New to Python or choosing between Python 2 and Python 3? Read Python 2 or Python 3.

The Python Software Foundation holds the intellectual property rights behind Python, underwrites the PyCon conference, and funds other projects in the Python community.

Read more, or download Python now

- **PyArkansas**
  The 4th annual PyArkansas will be held October 22, 2011.
  Published: Wed, 21 September 2011, 11:37 +0200

- **PyGotham**
  PyGotham will be held September 16-17, 2011.
  Published: Tue, 6 September 2011, 08:00 +0200

http://www.python.org
Python and ArcGIS

• Python is the preferred scripting language for ArcGIS

1. You can run Python from within ArcGIS
   – Python Window works like an interactive interpreter

2. All tools in ArcToolbox can be accessed from Python
   – Import ArcPy to get full library of tools

3. Python scripts can be made into tools
   – Extend functionality of ArcGIS

4. Support for other scripting languages will go away
   – VBScript and JScript being replaced by Python
Python Versions and ArcGIS

• Versions:
  – Current version of Python is 3.2.2
  – Python that works with ArcGIS 10.0 is 2.6.x
  – Python that works with ArcGIS 10.1 is 2.7.x
  – Move to Python 3.x likely only with ArcGIS 11

• ArcGIS only works with a specific version of Python:
  – Use the one that comes installed with ArcGIS
  – Don’t install your own version of Python
Installing Python

• Remove any existing installations of Python

• Install ArcGIS 10.0
  – Python 2.6.5 will be installed by default

• Install a Python editor

• Configure the editor to work with ArcGIS

• Note: You can run different versions of Python on one machine – however, a clean install of Python 2.6.5 with ArcGIS 10.0 is recommended
Demo: Check ArcGIS and Python installation
Fundamentals of Geoprocessing in ArcGIS
Geoprocessing Tools
Tool Organization
Tool Dialogs
Tool Parameters

• Parameters
  – Required
  – Optional

• Errors

• Warning
Environment Settings

- Workspace
- Output Coordinates
- Processing Extent
- XY Resolution and Tolerance
- M Values
- Z Values
- Geodatabase
- Geodatabase Advanced
- Fields
- Random Numbers
- Cartography
- Coverage
- Raster Analysis
- Raster Storage
- Geostatistical Analysis
- Terrain Dataset
- TIN
Demo: Geoprocessing Fundamentals
Running Python Code
Two ways to run Python Code

1. Using an Interactive Interpreter
   – Code is executed directly line-by-line

2. By running a script
   – Code saved in a .py file
   – Run from within a Python editor or directly from operating system
Where to type and run Python code?

1. Python window in ArcGIS
   – Built into any ArcGIS Desktop application
   – Good for testing code, very short scripts

2. Python editor
   – IDLE installed by default
   – Many others, PythonWin is a good one to start
   – Good for more complex code, saving scripts
Python Window in ArcGIS

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

>>> }
```
Python Window in ArcGIS

- Works with current map document
- Interactive interpreter:
  - Executes code directly line-by-line
- Good for testing short code
- Code can be saved
- No error checking / debugging
Python Editor - IDLE

Python 2.6.5 (r265:79096, Mar 19 2010, 21:48:26) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.

********************************************************************************
Personal firewall software may warn about the connection IDLE
makes to its subprocess using this computer's internal loopback
interface. This connection is not visible on any external
interface and no data is sent to or received from the Internet.
********************************************************************************

IDLE 2.6.5
>>>
Python Editor - PythonWin

```
>>> Hello World

print "Hello World"
```

Script 'C:\data\hellowin.py' returned exit code 0
Python Editor

• Stand-alone – outside of ArcGIS
• Interactive interpreter:
  – Executes code directly line-by-line
• Save code as script files (.py)
• Good for organizing more complex code
Demo: Running simple Python code
Python Documentation
Python Programming Language – Official Website

Python is a programming language that lets you work more quickly and integrate your systems more effectively. You can learn to use Python and see almost immediate gains in productivity and lower maintenance costs.

Python runs on Windows, Linux/Unix, Mac OS X, and has been ported to the Java and .NET virtual machines.

Python is free to use, even for commercial products, because of its OSI-approved open source license.

New to Python or choosing between Python 2 and Python 3? Read Python 2 or Python 3.

The Python Software Foundation holds the intellectual property rights behind Python, underwrites the PyCon conference, and funds other projects in the Python community.

Read more, or: download Python now

» PyArkansas
The 4th annual PyArkansas will be held October 22, 2011.
Published: Wed, 21 September 2011, 11:37 +0200

» PyGotham
PyGotham will be held September 16-17, 2011.
Published: Tue, 6 September 2011, 08:00 +0200

http://www.python.org
Python Documentation

Python v2.6.7 documentation

Welcome! This is the documentation for Python 2.6.7, last updated Jun 03, 2011.

Parts of the documentation:

- What's new in Python 2.6? or all "What's new" documents since 2.0
- Tutorial
  start here
- Using Python
  how to use Python on different platforms
- Library Reference
  keep this under your pillow
- Language Reference
  describes syntax and language elements
- Python HOWTOs
  in-depth documents on specific topics
- Extending and Embedding
  tutorial for C/C++ programmers
- Python/C API
  reference for C/C++ programmers
- Installing Python Modules
  information for installers & sys-admins
- Distributing Python Modules
  sharing modules with others
- Documenting Python
  guide for documentation authors
- FAQs
  frequently asked questions (with answers!)

http://docs.python.org
Python Beginners Guide

Beginner's Guide to Python

New to programming? Python is free, and easy to learn if you know where to start! This guide will help you to get started quickly.

Chinese Translation

New to Python?

Read BeginnersGuide/Overview for a short explanation of what Python is.

Getting Python

Next, install the Python interpreter on your computer. This is the program that reads Python programs and carries out their instructions; you need it before you can do any Python programming.

There are currently two major versions of Python available: Python 2 and Python 3. The Python2orPython3 page provides advice on how to decide which one will best suit your needs. At the time of writing (21 Jun 2010), the rest of this page assumes you've decided to use Python 2.

See BeginnersGuide/Download for instructions for downloading the correct version of Python.

At some stage, you'll want to edit and save your program code. Take a look at HowToEditPythonCode for some advice and recommendations.

http://wiki.python.org/moin/BeginnersGuide
Python Books

Version specific!

None of these books including anything on ArcGIS or geoprocessing!
Python Language Fundamentals
Python Data Types

- Number (integer and float)
- String
- List
- Tuple
- Dictionary

- Strings, lists and tuples are *sequences*
- Strings, numbers and tuples are *immutable*
- List and dictionaries are *mutable*
Numbers

• Integers
  – Whole number, i.e. no decimals
  – e.g. -34

• Floats
  – Decimal point
  – e.g. -34.8307
## Numerical Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Integer</th>
<th>Floating-point</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Example</td>
<td>Result</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
<td>9 * 2</td>
<td>18</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
<td>9 / 2</td>
<td>4</td>
</tr>
<tr>
<td>%</td>
<td>Modulus</td>
<td>9 % 2</td>
<td>1</td>
</tr>
<tr>
<td>+</td>
<td>Addition</td>
<td>9 + 2</td>
<td>11</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
<td>9 - 2</td>
<td>7</td>
</tr>
</tbody>
</table>
Demo: Numerical Operators
Strings

• A set of characters surrounded by quotes is called a *string literal*

• To create a *string variable*, assign a string literal to it

```python
>>> mytext = "Crime hotspot maps are cool."
>>> print mytext
Crime hotspot maps are cool.
```
Quotes in Python

• In Python single and double quotes are the same
• "NIJ" is the same as 'NIJ'

```python
>>> print "I said: 'Let's go!'"
```

• Quotes in Python are straight-up
• "text" or 'text', not “text” or ‘text’

• Be aware of copy/paste and auto-formatting
Variables

• Python scripts use *variables* to store information
• To work with variables use an *assignment* statement

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

• Python uses *dynamic* assignment

```python
>>> x = 17
>>> type(x)
<type 'int'>
>>> x = "GIS"
>>> type(x)
<type 'str'>
```

• No need to declare variables
• Value defines the type
Variable Names

• Rules
  – Letters, digits and underscores
  – Cannot start with a digit
  – Don’t use keywords (\texttt{print, import, etc.})

• Recommendations
  – Be descriptive (\texttt{count instead of c})
  – Keep it short (\texttt{count instead of count\_of\_records})
  – Follow convention: all lowercase, use underscores
Statement and Expressions

• A Python *expression* is a value

```python
>>> 2 * 17
34
```

• A Python *statement* is an instruction to do something

```python
>>> x = 2 * 17
```
Working with Strings

• Concatenate strings

```python
>>> x = "G"
>>> y = "I"
>>> z = "S"
>>> print x + y + z
GIS
```
Converting to String

```python
>>> temp = 100
>>> print "The temperature is " + temp + " degrees"
TypeError: cannot concatenate 'str' and 'int' objects

>>> print "The temperature is " + str(temp) + " degrees"
```

- Converting the value of a variable from one type to another is known as *casting*
Lists

- A Python list is an ordered set of items
- The list of items is surrounded by square brackets [ ], and the items are separated by commas (,)
- Items can consist of numbers, strings and other data types

mylist = [1, 2, 4, 8, 16, 32]
mywords = ["jpg", "bmp", "tif"]

- Lists are very widely used in geoprocessing:
  - e.g. list of feature classes, list of records, list of fields, etc.
Python Functions

• A *function* carries out a certain action
• Python has many built-in functions

```python
<function>(<arguments>)
```

```python
>> pow(2,3)
8
```

• Using a function is referred to as *calling* a function
• Additional functions can be accessed using *modules*
Python Methods

• A *method* is a function that is closely coupled to some object

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

```python
>>> topic = "Crime Mapping"
>>> topic . count("i")
2
```

• Many of Python’s data types have methods
String Indexing

- Python strings have an index positioning system

```python
>>> mystring = "Crime Mapping"
>>> mystring[0]
'C'
>>> mystring[-1]
'g'
```

- Strings can be sliced into smaller strings using *slicing*

```python
>>> mystring[0:5]
'Crime'
```
Working with List

- Python lists have an index positioning system

```python
>>> crimes = ['arson', 'burglary', 'robbery']
>>> cities[1]
'burglary'
```

- There are many list methods

```python
>>> crimes.append('homicide')
>>> crimes.remove('arson')
>>> crimes
['burglary', 'robbery', 'homicide']
```
Working with Pathnames

- Pathnames are critical when writing scripts:
  - Example workspace: c:\data\results
  - Example shapefile: c:\data\results\streams.shp
- In Python a backslash (\) is an escape character
- Pathnames in Python should therefore look like one of the following

"c:/data"
"c:\\data"
r"c:\data" (raw string)
Python Modules

- *Modules* are like extensions that can be imported into Python to extend its capabilities

```python
>>> import time

>>> time.localtime()
```
Conditional Statements

• *Branching* can be used to control workflow

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

• Syntax: keyword *if*, followed by a condition, followed by (:)

Indentation in Python

• Indented code is referred to as a *block*
• Use tabs or spaces – be consistent
• Recommended: 4 spaces

• *Tip: be careful with copy/paste from other applications*
More Conditional Statements

- Use of `elif` and `else` is optional

```python
import random
x = random.randint(0, 6)
print x
if x == 6:
    print "You win!"
elif x == 5:
    print "Try again!"
else:
    print "You lose!"
```
Loop Structures: While

- *Loop* structures allow you to repeat a certain part of your code
- A *while* loop repeats until a particular condition is reached

```python
i = 0
while i <= 10:
    print i
    i += 1
```

- The *while* statement uses a *sentry variable* in the exit condition
Loop Structures: For

- A `for` loop repeats a block of code for each element of a sequence

```python
mylist = ["A", "B", "C", "D"]
for letter in mylist:
    print letter
```

- In the example, `letter` is the name of a variable and for each iteration of the loop this variable is assigned a different value
ArcPy: Geoprocessing using Python
What is ArcPy?

• ArcPy was introduced with ArcGIS 10.0
• ArcPy is a collection of modules, classes and functions which give access to all the geoprocessing tools in ArcGIS from within Python
• Most geoprocessing scripts will start with:

```python
import arcpy
```

• *Note: ArcPy replaces the older* `arcgis scripting` *module*
Setting Current Workspace

- After importing ArcPy, most scripts start with setting a workspace to retrieve and store files

```python
import arcpy
arcpy.env.workspace = "c:/workshop"
```

- In the code above `env` is a class and `workspace` is a property of this class

```python
arcpy.<class>..<property>
```
Using Tools

• ArcPy gives you access to all tools in ArcToolbox
• All tools are provided as functions

\[
\text{arcpy.<toolname_toolboxalias>\(\langle\text{parameters}\rangle\)}
\]

• Example:

```python
import arcpy
arcpy.env.workspace = "c:/data"
arcpy.Clip_analysis("streams.shp", "study.shp", "result.shp")
```
Tool Parameters

- A good understanding of tool parameters is essential
- Parameters have properties:
  - Name
  - Type (feature class, integer, etc.)
  - Direction (input or output)
  - Required or optional
- Example: Clip tool

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>in_features</td>
<td>The features to be clipped.</td>
<td>Feature Layer</td>
</tr>
<tr>
<td>clip_features</td>
<td>The features used to clip the input features.</td>
<td>Feature Layer</td>
</tr>
<tr>
<td>out_feature_class</td>
<td>The feature class to be created.</td>
<td>Feature Class</td>
</tr>
<tr>
<td>cluster_tolerance</td>
<td>The minimum distance separating all feature coordinates (nodes and vertices) as well as the distance a coordinate can move in X or Y (or both). Set the value to be higher for data with less coordinate accuracy and lower for data with extremely high accuracy.</td>
<td>Linear unit</td>
</tr>
</tbody>
</table>
Tool Syntax

Tool dialog:

Python syntax:  
```
Clip_analysis(in_features, clip_features, 
out_feature_class, 
{cluster_tolerance})
```

Example:  
```
Clip_analysis("streams.shp","study.shp", 
"result.shp")
```
Optional Parameters

- Required tool parameters are listed first
- Optional tool parameters can be left out
  - But what if some need to be set?

```python
Buffer_analysis (in_features, out_feature_class
  buffer_distance_or_field, {line_side}, {line_end_type},
  {dissolve_option}, {dissolve_field})

arcpy.Buffer_analysis("roads", "buffer", "100 METERS", ", ",
  ", "LIST", "Code")

arcpy.Buffer_analysis("roads", "buffer", "100 METERS", 
dissolve_option=LIST, dissolve_field=Code)
```
Hard-coded Parameters

• Consider the example

```python
import arcpy
arcpy.env.workspace = "c:/data"
arcpy.Clip_analysis("streams.shp", "study.shp", "result.shp")
```

• How can we make this code more usable?
Using Variables for Parameters

```python
import arcpy
arcpy.env.workspace = "c:/data"
infc = "streams.shp"
clipfc = "study.shp"
outfc = "result.shp"
arcpy.Clip_analysis(infc, clipfc, outfc)
```
Variables Provided by a User

```python
import arcpy
infc = arcpy.GetParameterAsText(0)
clipfc = arcpy.GetParameterAsText(1)
outfc = arcpy.GetParameterAsText(2)
arcpy.Clip_analysis(infc, clipfc, outfc)
```
Result Objects

- ArcPy returns the output of a tool as a Result object

```python
import arcpy
arcpy.env.workspace = "c:/data"
myresult = arcpy.Clip_analysis("streams.shp","study.shp","result.shp")
print myresult
```

- This will print the path to the output dataset

```
c:/data/result.shp
```
Multiple Operations using Result Objects

• Result objects can be used as the input into another function

```python
import arcpy
arcpy.env.workspace = "c:/data/study.gdb"
buffer = arcpy.Buffer_analysis("str","str_buf","100 METERS")
count = arcpy.GetCount_management(buffer)
print count
```

• This allows complex geoprocessing operations
ArcPy Classes

• Some tool parameters are complicated/detailed
  – e.g. coordinate system

• ArcPy classes are used to work with these parameters
  – Classes are used to create objects
  – Classes have properties and methods

• General syntax

  arcpy.<classname>(<parameters>)
ArcPy Classes: Example

• The following is an example of the contents of a .prj file

![Code snippet showing a .prj file content]

• To avoid having to work with this actual string, we can use a `SpatialReference` class
import arcpy
prjfile = "c:/data/streams.prj"
spatialref = arcpy.SpatialReference(prjfile)
myref = spatialRef.name
print myRef

• This will print

NAD_1983_StatePlane_Florida_East_FIPS_0901_Feet
ArcPy Classes: Example

- The following example creates a spatial reference object and use this to define the coordinate system of a new feature class

```python
import arcpy
out_path = "c:/data"
out_name = "lines.shp"
prjfile = "c:/data/streams.prj"
spatialref = arcpy.SpatialReference(prjfile)
arcpy.CreateFeatureclass_management(out_path, out_name,
   "POLYLINE", ",", ",", ",", spatialref)
```
ArcPy Functions

- All geoprocessing tools are ArcPy functions
- Additional ArcPy functions:
  - listing data
  - Retrieving and setting properties
  - Many more...
- General syntax

\[ \text{arcpy.<functionname>}(\text{<arguments>}) \]
ArcPy Functions

- Cursors
- Describing data
- Environment and settings
- Fields
- General
- General data functions
- Getting and setting parameters
- Licensing and installation
- Listing data
- Messaging and error handling
- Progress dialog
- Tools and toolboxes
Describing and Listing Data
Describing Data

- The Describe function is used to determine properties of dataset
- General syntax

```python
import arcpy
<variable> = arcpy.Describe(<input dataset>)
```

- Example:

```python
import arcpy
desc = arcpy.Describe("c:/data/streams.shp")
print desc.shapeType
```
import arcpy
arcpy.env.workspace = "c:/data"
inf = "streams.shp"
clipf = "study.shp"
outf = "streams_clip.shp"
desc = arcpy.Describe(clipf)
type = desc.shapeType
if type == "Polygon":
    arcpy.Clip_analysis(infc, clipf, outf)
else:
    print "The clip features are not polygons."
Listing Data

• Listing data is very common
• Several different list functions in ArcPy
  – ListFields
  – ListIndexes
  – ListDataset
  – ListFeatureClasses
  – ListFiles
  – ListRasters
  – ListTables
  – ListWorkspaces
  – ListVersions
• Similar logic:
  – Create a list
  – Iterate over the list using a for loop
Listing Feature Classes

- The `ListFeatureClasses` function returns a list of feature classes in the current workspace
- General syntax:

  ```python
  ListFeatureClasses ({wild_card}, {feature_type}, {feature_dataset})
  ```

- Example:

  ```python
  import arcpy
  from arcpy import env
  env.workspace = "c:/data"
  fclist = arcpy.ListFeatureClasses()
  ```
Listing Feature Classes

- No filtering:

```python
fclist = arcpy.ListFeatureClasses()
```

- Filtering based on wild card

```python
fclist = arcpy.ListFeatureClasses("w*")
```

- Filtering based on feature type

```python
fclist = arcpy.ListFeatureClasses("", "point")
```
Listing Fields

- The **ListFields** function lists the fields in a feature class or table in a specified dataset.

- **General syntax:**

  ```python
  ListFields (dataset, {wild_card}, {field_type})
  ```

- **Example**

  ```python
  import arcpy
  arcpy.env.workspace = "c:/data"
  fieldlist = arcpy.ListFields("roads.shp")
  ```
Using Lists in for loops

• The following script creates a list of fields of type String and determines for each text field what the length of the field is

```python
import arcpy
arcpy.env.workspace = "c:/data"
fieldlist = arcpy.ListFields("roads.shp", "", "String")
for field in fieldlist:
    print field.name + " " + str(field.length)
```
Using Lists in for loops

• The following script creates a list of TIFF files and iterates through each file in the list to build pyramids

```python
import arcpy
from arcpy import env
env.workspace = "c:/data"
tifflist = arcpy.ListRasters("","TIF")
for tiff in tifflist:
    arcpy.BuildParamids_management(tiff)
```
Creating Custom Tools
Ways to Execute a Script

1. As a stand-alone script
   - The script is executed from the operating system or from within a Python editor such as PythonWin
   - When using ArcPy, ArcGIS needs to be installed and licensed
   - No ArcGIS Desktop application needs to be open

2. As a script tool within ArcGIS
   - A tool dialog is created to execute the script
   - Script tool looks like any other tool in ArcToolbox
   - Tool execution is controlled from ArcGIS Desktop
Python Scripts as Tools

```python
# HuffModel.py
# Created: 4/13/2007 by Drew Rinter
// Usage: Creating probability-based trade areas for retail stores
// -------------------------------------------------------------

# Import system modules
import sys, string, arcgisscripting, os, traceback, shutil, re

# Create the Geoprocessor object
gp = arcgisscripting.create()  

# Set overwrite
gp.overwriteoutput = 1

def AddPrintMessage(msg, severity):
    print msg
    if severity == 0: gp.AddMessage(msg)
    elif severity == 1: gp.AddWarning(msg)
    elif severity == 2: gp.AddError(msg)

# Start traceback Try-Except statement:
try:
    # Script parameters...
    stores = gp.getparameterasertext(0)
    store_name = gp.getparameterasertext(1)
    store_attr = gp.getparameterasertext(2)
    output_folder = gp.getparameterasertext(3)
    feat_name = gp.getparameterasertext(4)
    studyarea = gp.getparameterasertext(5)
    blockgroups = gp.getparameterasertext(6)
```
Why Create Script Tools?

- Tool dialog makes it easier to use
- Tool dialog validates user inputs
- Becomes part of all geoprocessing
- Environment settings are passed on
- Writes messages to the Results window
- Easier to share
- Does not require user to know Python
Steps to Create Script Tools

1. Create a Python script (.py)
2. Create a custom Toolbox (.tbx)
3. Add a tool to the Toolbox using Add Script
4. Modify the script with inputs and outputs
Example Script: Hardcoded Variables

```python
import arcpy
from arcpy import env
env.overwriteoutput = True
infc = "c:/data/points.shp"
output = "c:/data/result.txt"
k = 10
n = 1
f = open(output, "w")
while n <= k:
    result = arcpy.CalculateDistanceBand_stats(infc, n)
f.write(str(n) + " " + str(result[1]) + "\n")
n = n + 1
f.close()
```
Tool Parameters and Dialog

K-Nearest Neighbor Properties

Parameter Properties:
- Property: Value
- Type
- Direction
- Multi Value
- Default
- Environment

To add a new parameter, type the name into an empty row in the name column, click in the Data Type column to choose a data type, then edit the Parameter Properties.
import arcpy
from arcpy import env
ev.env.overwriteoutput = True
infc = arcpy.GetParameterAsText(0)
output = arcpy.GetParameterAsText(1)
k = arcpy.GetParameter(2)
n = 1
f = open(output, "w")
while n <= k:
    result = arcpy.CalculateDistanceBand_stats(infc, n)
f.write(str(n) + " " + str(result[1])+ "\n")
n = n + 1
f.close()
More ArcPy Functionality
More ArcPy Functionality

- Cursors to work with rows and geometry
  - Retrieve, edit, create
- `arcpy.sa` module to work with rasters
- `arcpy.mapping` module for map automation
- Creating custom functions and classes
Resources for Python Scripting in ArcGIS
ArcPy is a site-package that builds on (and is a successor to) the successful arcgisscripting module. Its goal is to create the cornerstone for a useful and productive way to perform geographic data analysis, data conversion, data management, and map automation with Python.

This package provides a rich and native Python experience offering code completion (type a keyword and a dot to get a pop-up list of properties and methods supported by that keyword; select one to insert it) and reference documentation for each function, module, and class.

The additional power of using ArcPy within Python is the fact that Python is a general purpose programming language. It is interpreted and dynamically typed and is suited for interactive work and quick prototyping of one-off programs known as scripts while being powerful enough to write large applications in. ArcGIS applications written with ArcPy benefit from the development of additional modules in numerous niches of Python by GIS professionals and programmers from many different disciplines.

General Help

Python provides the facility of documentation strings. The functions and classes available in ArcPy use this method for the package documentation. One method for reading these messages and getting help is by using the command help provided by Python. Running the command with an argument displays the calling signature and the documentation string of the object.

```python
>>> import arcpy
>>> help(arcpy)
```
Virtual Campus Courses

Using Python in ArcGIS Desktop 10

Description
At ArcGIS Desktop 10, Python scripting is tightly integrated into ArcMap and ArcCatalog, allowing you to create and automate GIS workflows quickly and easily. This course introduces Python scripting in ArcGIS Desktop and shows how you can use scripts to increase productivity and the quality of your maps and data. The presentation covers how to use the new ArcPy mapping module to manipulate map documents and layers.

Who Should Attend
GIS analysts, specialists, and other experienced ArcGIS users who want to automate complex tasks and common procedures.

Goals
After completing this course, you will be able to

- Create basic Python scripts using correct syntax.
- Write and run scripts in ArcMap using the Python window.
- Use Python in the Field Calculator.
- Create script tools to automate geoprocessing operations.

http://training.esri.com
ArcScripts

http://arcscripts.esri.com
ArcGIS Resource Center

http://resources.arcgis.com
Modeling and analysis

Spatial analysis is one of the more interesting and remarkable aspects of GIS. Using spatial analysis, you can combine information from many independent sources and derive a new set of information (results)—by applying a large, rich, and sophisticated set of spatial operators. GIS professionals use geoprocessing to program their own ideas in order to derive these analytical results. In turn, these results are applied to a wide variety of problems. For example, here, geoprocessing is used to identify suitable sites for parks. The result is a dataset of potential park sites for further evaluation. Site selection logic is used to find areas that are close to where people live but are not too close to existing parks.

Learn more about ModelBuilder

http://resources.arcgis.com/content/geoprocessing/10.0/about
Beyond ArcGIS
Using PySAL for Spatial Analysis

http://geodacenter.asu.edu/pysal
PySAL

• Python library of spatial analysis methods
• ESDA, spatial statistics, geostatistics
• Growing and expandable
Using R for Spatial Analysis

- Open source language for data analysis
- Libraries have been developed for spatial methods
- Large and active user community
- Growing and expandable
ArcGIS and R
Script Tool
Python script that calls R

```python
import arcpy as ARCPY
import arcpy.management as M
import os as OS
import sys as SYS
import subprocess as SUB

### Parameter Dictionaries ###
clusterDict = {"KMEANS_HARTIGAN": "kmeansHartigan", "CLARA": "clara", "R_CLUST": "cluster", "R_CLUST": "kmeans", "R_CLUST": "kmeans", "R_CLUST": "kmeans", "R_CLUST": "kmeans3"}

def PointClusters():
    ### Get User Provided Inputs ###
    inputFC = "\" + ARCPY.GetParameterAsText(0) + "\"
    masterField = str(ARCPY.GetParameterAsText(1))
    outputFC = "\" + ARCPY.GetParameterAsText(2) + "\"
    numClusters = ARCPY.GetParameterAsText(3)
    clusterMethodStr = ARCPY.GetParameterAsText(4)
    vacNames = ARCPY.GetParameterAsText(5)
    vacNames = [str(i) for i in vacNames.split("\:\")]
    vacNames = ";".join(vacNames)

    ### Create R Command ###
    pyScript = SYS.argv[0]
    toolsDir = OS.path.dirname(pyScript)
    rScript = toolsDir + "/PointClusters.R"
    rScript = "\" + rScript + "\"
    ARCPY.SetProgressor("Default", "Executing R Script...")
    args = ";\".join([inputFC, masterField, outputFC, numClusters, clusterMethodStr, vacNames])
    cmd = RCMD + args + " < " + rScript

    ### Uncomment Next Line to Print Command ###
    #ARCPY.AddWarning(cmd)

    ### Execute Command ###
    OS.system(cmd)

    ### Project the Data ###
    OS.renameProjection(outputFC, strip("\"), inputFC, strip("\"))

    ### Render the Results ###
    params = ARCPY.spa.GetParameterInfo()
    renderFile = OS.path.join(toolDir, "RenderClusters.lyr")
    params[2].Symbolize = renderFile
```
Evaluating R Statements
Concluding Remarks

• Python is a relatively easy to learn language

• ArcGIS is becoming more “Pythonesque”

• Creating time-savings scripts for repetitive tasks does not take a lot of code

• Easy to share script tools
Paul Zandbergen

Department of Geography
zandberg@unm.edu
www.paulzandbergen.com
Workshop Materials Posted

http://www.paulzandbergen.com/workshops