2011 Esri Developer Summit
Palm Springs, CA

Python for Working with ArcGIS

Jason Pardy & Ghislain Prince

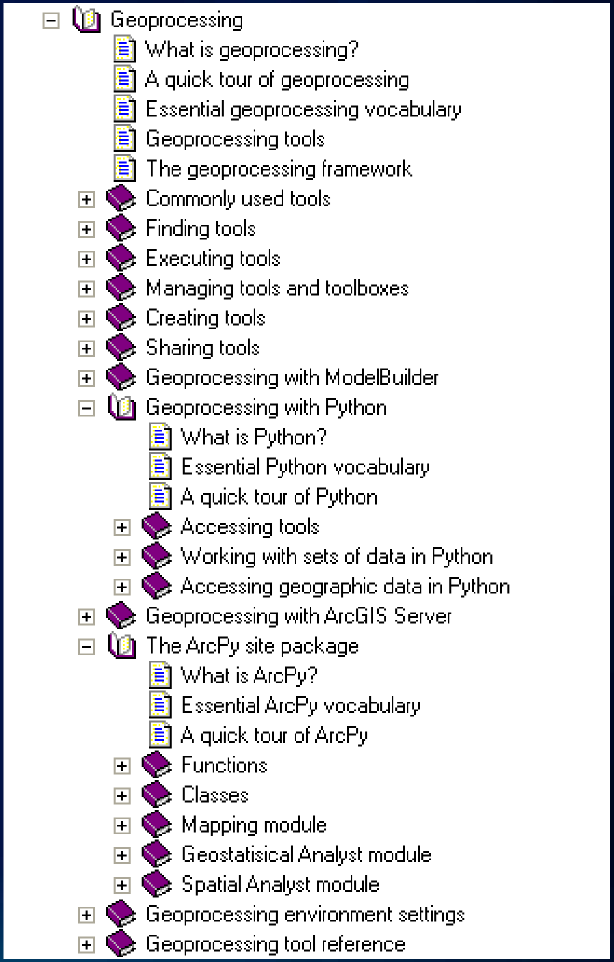
esri

Agenda

- Essentials
  - Why use Python scripting?
  - What is ArcPy?
  - Executing tools
  - Messages and error handling
  - ArcPy Classes
  - Cursors
- Automation
  - ArcPy functions
  - Batch processing
  - Map automation
- Creating Script tools
- Spatial Analyst Module

Learning Python Scripting with ArcGIS

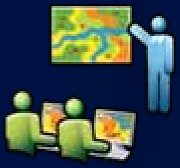
- Resource Center
  - http://resources.arcgis.com/geoprocessing/
- Desktop Help
- Have a good Python Reference
  - "Learning Python" by Mark Lutz
    - published by O'Reilly & Associates
  - "Core Python" by Wesley J. Chun
    - published by Prentice-Hall



- Geoprocessing
  - What is geoprocessing?
  - A quick tour of geoprocessing
  - Essential geoprocessing vocabulary
  - Geoprocessing tools
  - The geoprocessing framework
  - Commonly used tools
  - Finding tools
  - Executing tools
  - Managing tools and toolboxes
  - Creating tools
  - Sharing tools
  - Geoprocessing with ModelBuilder
  - Geoprocessing with Python
    - What is Python?
    - Essential Python vocabulary
    - A quick tour of Python
    - Accessing tools
    - Working with sets of data in Python
    - Accessing geographic data in Python
  - Geoprocessing with ArcGIS Server
  - The ArcPy site package
    - What is ArcPy?
    - Essential ArcPy vocabulary
    - A quick tour of ArcPy
    - Functions
    - Classes
    - Mapping module
    - Geostatisical Analyst module
    - Spatial Analyst module
  - Geoprocessing environment settings
  - Geoprocessing tool reference

Esri Training for Python

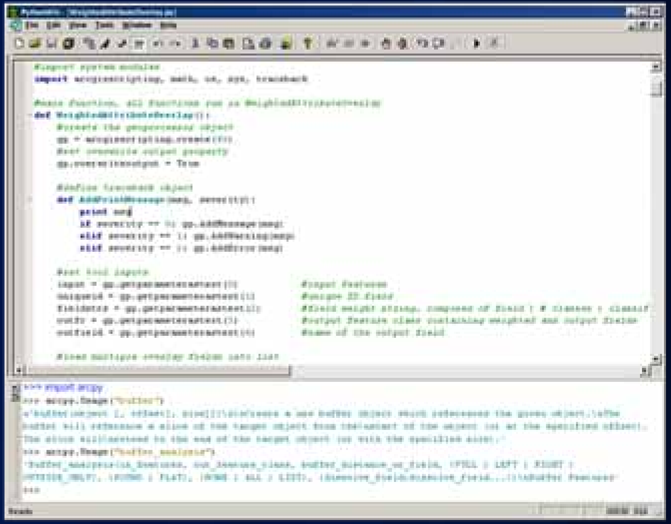
http://www.esri.com/training



- Instructor-Led Course
  - Introduction to Geoprocessing Scripts Using Python
- Web Course
  - Using Python in ArcGIS Desktop 10

Why Python?

- Fulfills the needs of our user community
  - Simple and easy to learn
  - Modular
  - Object oriented
  - Easy to maintain
  - Scalable
  - Cross platform (Windows & UNIX/Linux)
  - Established and active user community

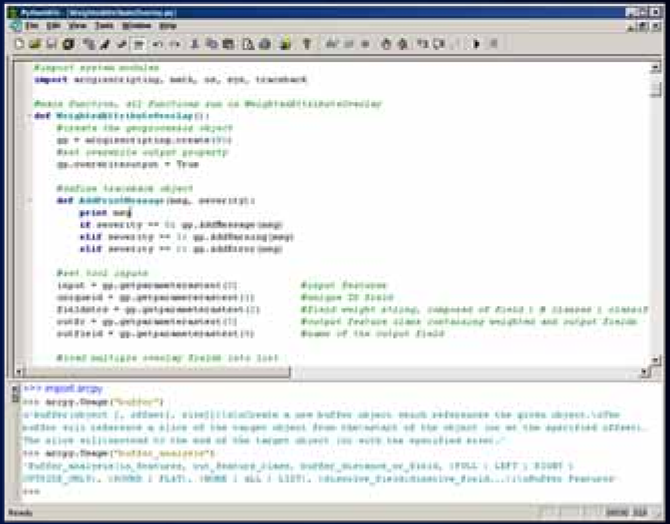


```
#import system modules
import arcgisscripting, math, os, sys, traceback

[illegible]
def [illegible]():
    [illegible]
    gp = arcgisscripting.create([illegible])
    [illegible]
    gp.overwriteoutput = True

    [illegible]
    def AddPrintMessage(msg, severity):
        print msg
        if severity == [illegible]: gp.AddMessage(msg)
        elif severity == [illegible]: gp.AddWarning(msg)
        elif severity == [illegible]: gp.AddError(msg)

    [illegible]
    input = gp.getparameterastext(0)         [illegible]
    [illegible] = gp.getparameterastext(1)   [illegible]
    [illegible] = gp.getparameterastext(2)   [illegible]
    [illegible] = gp.getparameterastext(3)   [illegible]
    outfield = gp.getparameterastext(4)      [illegible]

    [illegible]
```

```
>>> import arcpy
>>> arcpy.Usage("buffer")
[illegible]
>>>
```

Python is a productive language

- Significantly reduces the amount of time spent on a project.
  - Quickly execute tools or functions
  - Automate common tasks
- It is designed to be easy to read and learn
  - “Maintainability” – easy to modify and keep up to date



Scripting Fundamentals

- Provide an efficient method for defining and executing a workflow
- Create generic scripts that can be used multiple times
- Create new tools (analytical, data management, map production, etc.)

A brief history of Python in ArcGIS

9.0 / 9.1

- dispatch–based Geoprocessor
- *Python 2.1*

9.2

- arcgisscripting module
- Cross-platform
- *Python 2.4*

9.3

- arcgisscripting module, 9.3 version
- “Pythonic”
- *Python 2.5*

10.0

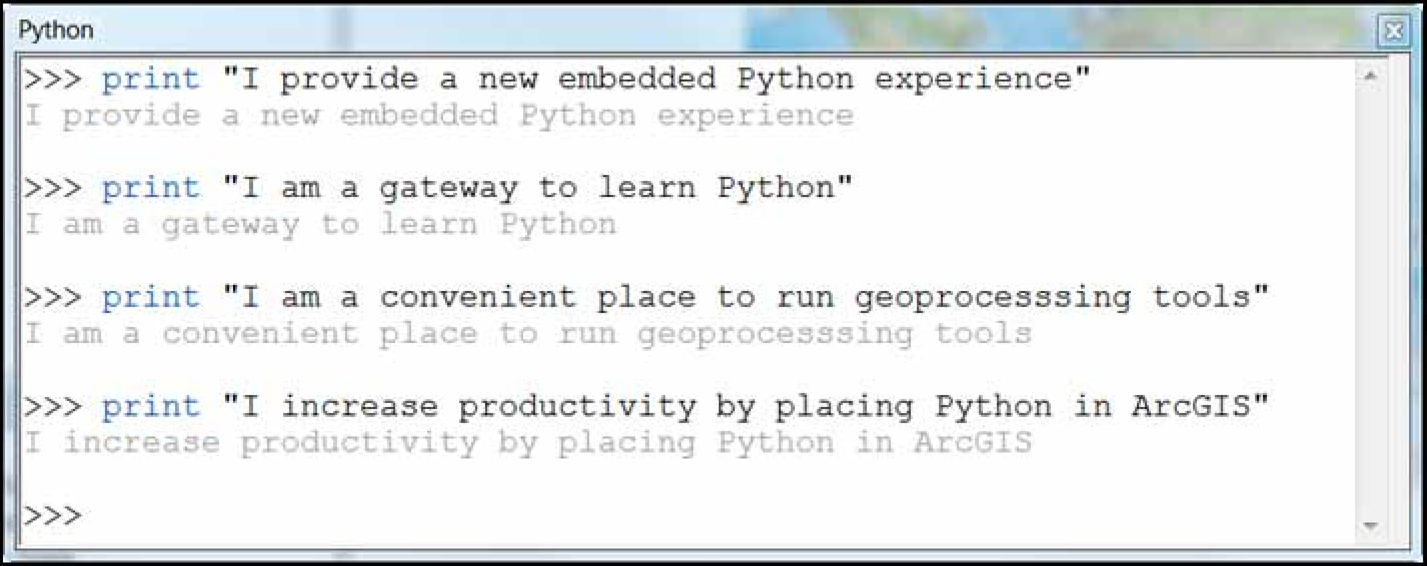
- ArcPy site-package
- Mapping & Map Algebra support
- Python window
- *Python 2.6*

What is ArcPy?

- **ArcPy is a native Python site-package**
- **Increases productivity with a richer and more native Python Experience**
- **Includes code completion and intellisense**
- **Includes modules covering other areas of ArcGIS:**
  - **Mapping**
  - **Extensions – Spatial Analyst (map algebra)**
- **Includes classes and functions making it easier to execute tools and create objects such as spatial references, geometries, etc.**

What is the Python window?

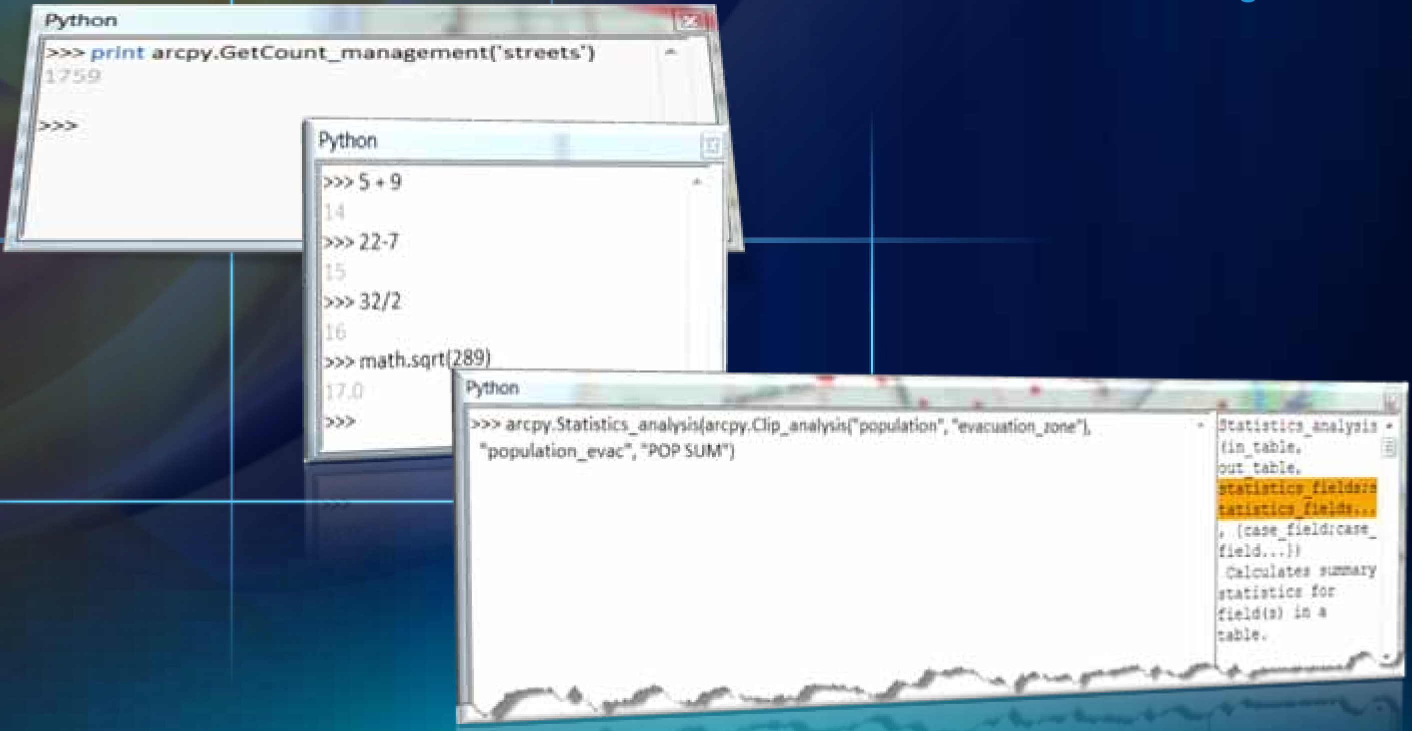
- An embedded Interactive Python window within ArcGIS
  - Can access ArcPy, including tools and environments
  - Can access any other Python functionality
  - Better code completion and intelligence



```
>>> print "I provide a new embedded Python experience"
I provide a new embedded Python experience

>>> print "I am a gateway to learn Python"
I am a gateway to learn Python

>>> print "I am a convenient place to run geoprocesssing tools"
I am a convenient place to run geoprocesssing tools

>>> print "I increase productivity by placing Python in ArcGIS"
I increase productivity by placing Python in ArcGIS

>>>
```

Demo

- Python window
- Executing Tools



```
>>> print arcpy.GetCount_management('streets')
1759

>>>
```

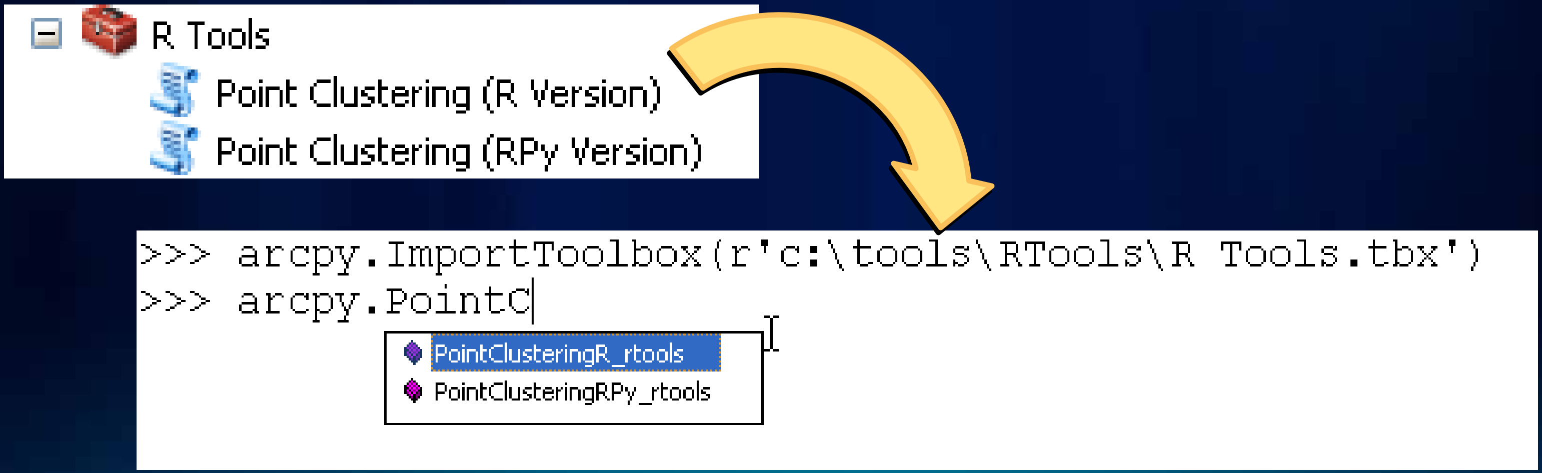
```
>>> 5 + 9
14
>>> 22-7
15
>>> 32/2
16
>>> math.sqrt(289)
17.0
>>>
```

```
>>> arcpy.Statistics_analysis(arcpy.Clip_analysis("population", "evacuation_zone"),
 "population_evac", "POP SUM")
```

```
Statistics_analysis
(in_table,
out_table,
statistics_fields;s
tatistics_fields...
, {case_field;case_
field...})
 Calculates summary
statistics for
field(s) in a
table.
```

Geoprocessing Tools

- Tools are the fundamental unit of geoprocessing
- There are hundreds of tools at your disposal
  - You can create your own tools (ModelBuilder, Python, etc.)
- Any tool, once created, can be called in Python by using the arcpy.ImportToolbox function
  - Creates tool wrappers for your toolbox


R Tools
Point Clustering (R Version)
Point Clustering (RPy Version)
PointClusteringR_rtools
PointClusteringRPy_rtools

```
>>> arcpy.ImportToolbox(r'c:\tools\RTools\R Tools.tbx')
>>> arcpy.PointC
```

Tool Messages

- Executing a tool will produce 3 types of messages.
  - Informative messages (severity = 0)
  - Warning messages (severity = 1)
  - Error messages (severity = 2)

```
# start try block
try:
    arcpy.Buffer("c:/ws/roads.shp", "c:/outws/roads10.shp", 10)

# If an error occurs when running a tool, print the tool messages
except arcpy.ExecuteError:
    print arcpy.GetMessages(2)

# Any other error
except Exception as e:
    print e.message
```

* A note on tool organization

- Tools can be accessed directly from arcpy

```
import arcpy
arcpy.GetCount_management(fc)
```

- Or from arcpy ‘toolbox’ modules

```
from arcpy.management import as dm
dm.GetCount(fc)
```

- *Matter of preference – functionally no difference*

Environments

- Script writers set the environment and tools use them
  - General settings
    - Current Workspace, Output Spatial Reference, Extent
  - Raster analysis settings
    - Cell Size, Mask
  - Many more

```
arcpy.env.workspace
arcpy.env.outputCoordinateSystem
arcpy.env.extent
arcpy.env.cellSize
```

Demo

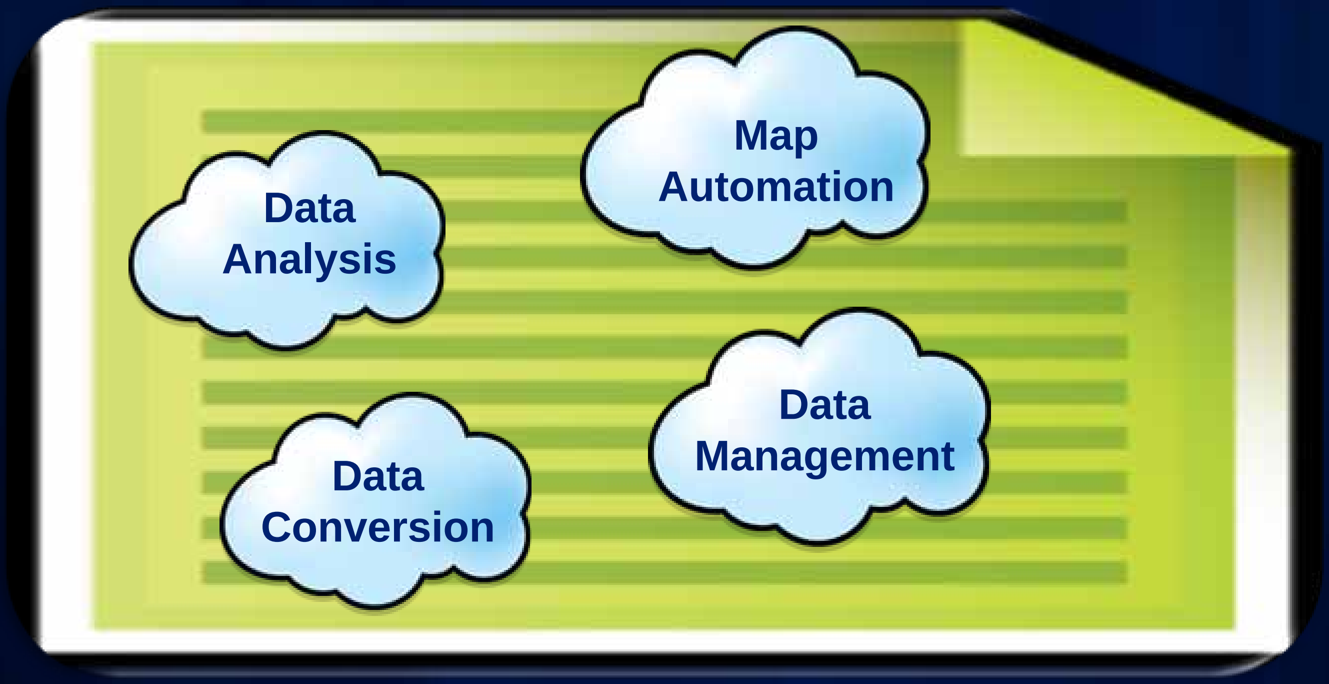
Setting Environments

Tool messages

Exception handling

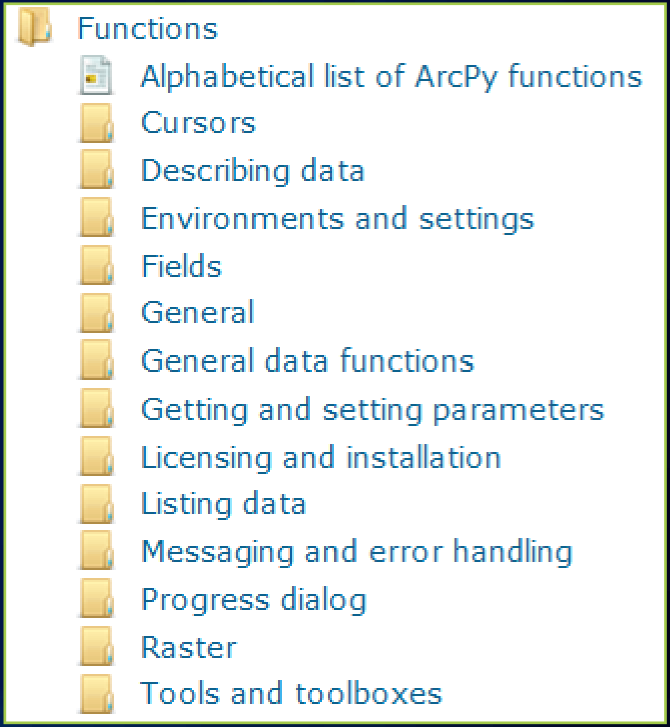
Automation = Productivity

Python extends across ArcGIS



Functions

- The ArcPy module contains functions necessary to perform many scripting tasks
  - Listing data
  - Describing data
  - Validating table and field names
  - Getting messages
  - etc.
- Allows automation of manual tasks

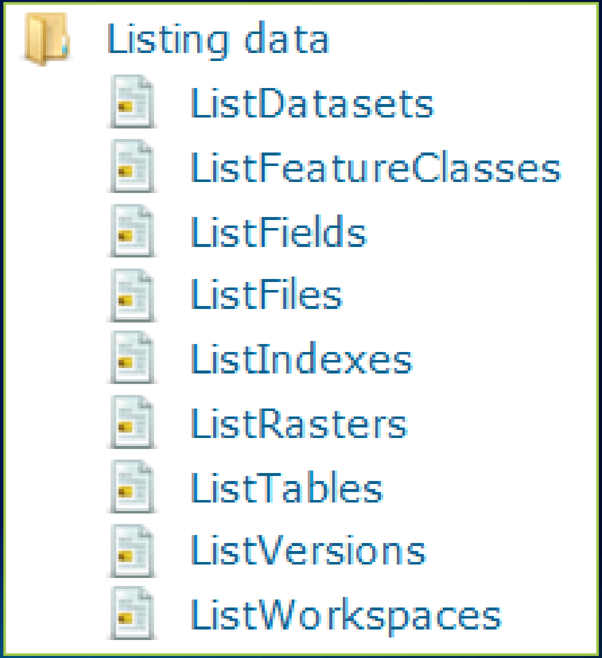


Functions
- Alphabetical list of ArcPy functions
- Cursors
- Describing data
- Environments and settings
- Fields
- General
- General data functions
- Getting and setting parameters
- Licensing and installation
- Listing data
- Messaging and error handling
- Progress dialog
- Raster
- Tools and toolboxes

Batch processing

- Geoprocessing tasks/jobs are often repeating on a set of data
  - Converting from one format to another (CAD to GDB)
  - Clipping a set of feature classes with a study area
  - Spill Modeling/Land use studies, etc.

- Several list methods exist to support these cases:



Listing data
- ListDatasets
- ListFeatureClasses
- ListFields
- ListFiles
- ListIndexes
- ListRasters
- ListTables
- ListVersions
- ListWorkspaces

Describing Data

- Allows script to determine properties of data
  - Data type (shapefile, coverage, network dataset, etc)
  - Shape type (point, polygon, line, etc)
  - Spatial reference
  - Extent of features
  - List of fields
- Returns an object with dynamic properties
- Logic can be added to a script to branch based on data properties

Demo

- Batch processing

Classes

- Most tool parameters can be easily defined
  - Such as a path or buffer distance
- Some parameters cannot be easily defined with a string
  - Such as a spatial reference or field mapping
- Classes can be used to define parameters

```
prjFile = "c:/North America Equidistant Conic.prj"

# Create a spatial reference using a projection file
spatialRef = arcpy.SpatialReference(prjFile)

# Run CreateFeatureclass using the spatial reference
arcpy.CreateFeatureclass_management(inputWorkspace,
    outputName, "POLYLINE", "", "", "", spatialRef)
```

Classes

- Classes can be used to more easily define *more involved* parameters
  - Such as a spatial reference or field mapping
- *No longer required to use CreateObject*

At 9.3

pt = gp.createObject("Point")
pt.x = 5
pt.y = 10

At 10
pt = arcpy.Point(5,10)

Accessing Data with Cursors

- Cursors can be used to iterate over the set of rows or insert new rows into a table
- Cursors are a workhorse for many workflows

Type	Explanation
SearchCursor	Read-only access
UpdateCursor	Update or delete rows
InsertCursor	Insert rows

Cursors

- **ArcPy cursors support iteration**

At 9.3

```
rows = gp.SearchCursor(myTable)
row = rows.next()
while row:
    print row.GetValue("Rank")
    row = rows.next()
```

At 10

```
for row in arcpy.SearchCursor(myTable)
    print row.GetValue("Rank")
```

Cursors

- Need coordinate information in a different coordinate system?
- Features may be projected on-the-fly using the Spatial Reference parameter

```
# Create a SR object from a projection file
SR = arcpy.SpatialReference("c:/NAD 1983 UTM Zone 10N.prj”)

# Create search cursor, using spatial reference
rows = arcpy.SearchCursor("D:/data.mdb/roads","", SR)
```

Accessing geometry with Cursors

- Feature classes have a geometry field
  - Typically *(but not always)* named Shape
- A geometry field returns a geometry object
- Geometry objects have properties that describe a feature
  - area, length, isMultipart, partCount, pointCount, type, …

```
# Find the total length of all line features
import arcpy
length = 0
for row in arcpy.SearchCursor("C:/data/base.gdb/roads"):
   feature = row.shape
   length += feature.length
```

Reading Feature Geometry

- You must understand the hierarchy for geometry in order to use it
  - A feature class is made of features
  - A feature is made of parts
  - A part is made of points
- In Python terms
  - A single part feature looks like this

    [pnt, pnt, pnt]
  - A multipart polygon feature looks like this

    [[pnt, pnt, pnt],[pnt, pnt, pnt]]
  - A single part polygon feature with a hole (inner ring) looks like

    [[pnt, pnt, pnt, ,pnt, pnt, pnt]]

Reading Feature Geometry

```
for row in arcpy.SearchCursor(polygonFC):

  for part in row.shape:
    pnt = part.next()

    while pnt:
      print pnt.X, pnt.Y
      pnt = part.next()

      if not pnt:
        pnt = part.next()
        if pnt:
          interiorRing = True
```

Loop through each row

Loop through each part in a feature

Loop through each point in a part

For polygons, watch for interior rings

Writing Feature Geometry

- Insert cursors must be used to create new features

```
rows = arcpy.InsertCursor("D:/data.gdb/roads")
row = rows.newRow()
```

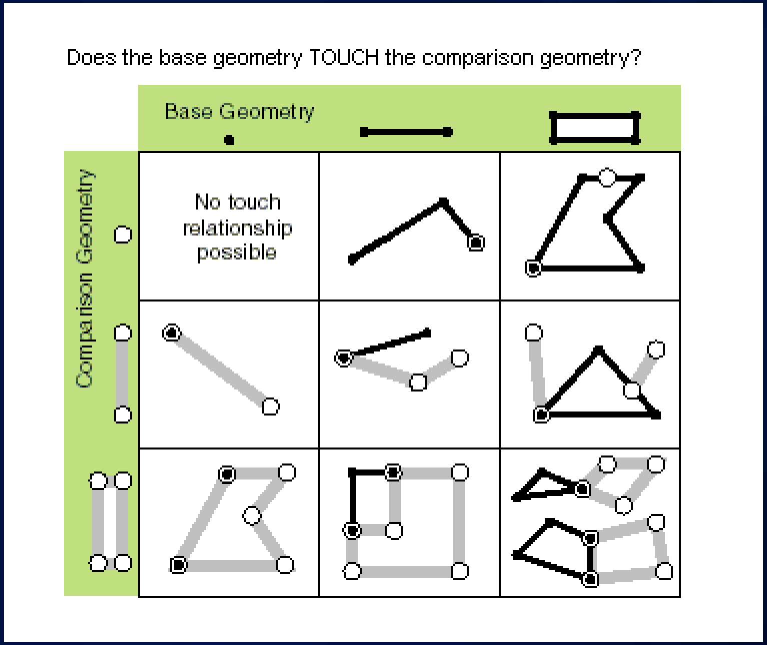
- Use Point and Array objects to create feature parts
- A part may be used to set a geometry field
  - A multipart feature is an array containing other arrays, where each array is a part
- An Update cursor can be used to replace a row’s existing geometry

Writing Feature Geometry

```
# Open an insert cursor for the feature class
cur = arcpy.InsertCursor(fc)

# Create array and point objects
ptList = [arcpy.Point(358331, 5273193),
              arcpy.Point(358337, 5272830)]

lineArray = arcpy.Array(ptList)

# Create a new row for the feature class
feat = cur.newRow()

# Set the geometry of the new feature to the array of points
feat.Shape = lineArray

# Insert the feature
cur.insertRow(feat)

# Delete objects
del cur, feat
```

Geometry operators

- Geometry objects support relational operators at 10
  - contains
  - crosses
  - disjoint
  - equals
  - overlaps
  - touches
  - within


Does the base geometry TOUCH the comparison geometry?
Base Geometry
Comparison Geometry
No touch relationship possible

Demo

cursor

features’ shape

relational operator

arcpy.mapping

mapping module

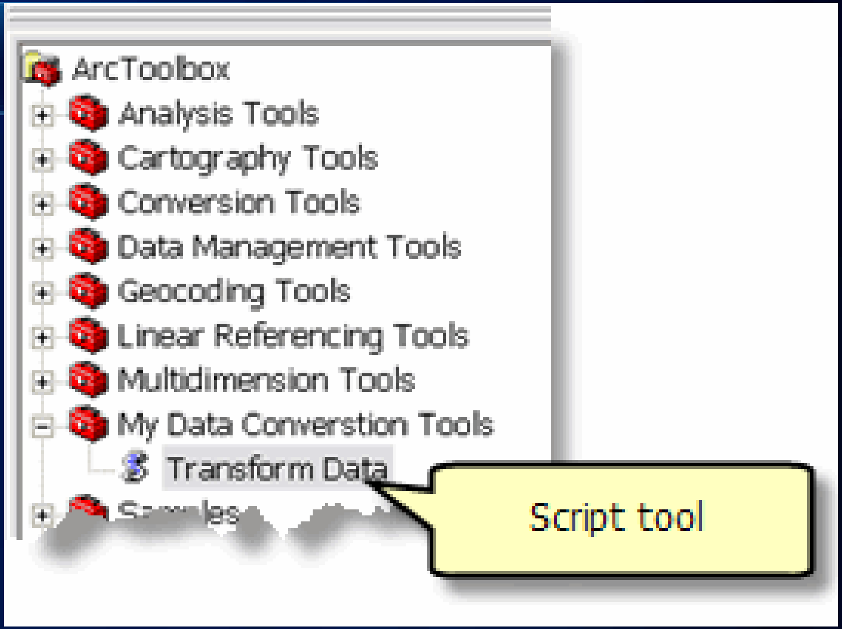
- A new mapping module that is part of the Geoprocessing ArcPy site-package
- A python scripting API that allows you to:
  - Manage map documents, layer files, and the data within
    - Find a layer with data source *X* and replace with *Y*
    - Update a layer's symbology across many MXDs
    - Generate reports that lists document information
      - Data sources, broken layers, spatial reference, info, etc.
  - Automate the exporting and printing of map documents
  - Automate map production/map series

Demo

Map automation

Script Tools

- Source is a script
- It is a tool
  - Use in ModelBuilder
  - Use in other scripts
  - “Full-fledged member”
- Since 9.3, runs in process
- Inherits all geoprocessing properties
- Communicates with application
  - Layers added to map, etc.
  - Messages
- More easily shared
  - Puts a familiar face on your work



Creating Tools from Scripts

- Why?
  - The script is generic and can be used with other data
    - Script can use arguments from the user
  - You want to use a script in ModelBuilder
  - Easier to share your script
    - Not everyone knows how to run a stand-alone script
  - Puts a familiar face on your work

Demo

Creating a script tool

Creating Tools from Scripts

- Step 1: Create argument variables
  - Use GetParameterAsText to obtain script argument values

- Step 2: Add messaging to your script
  - Return informative messages during execution of the script
  - Return error messages when a problem arises
  - Three functions to support tool messaging
    - AddMessage()
    - AddWarning()
    - AddError()

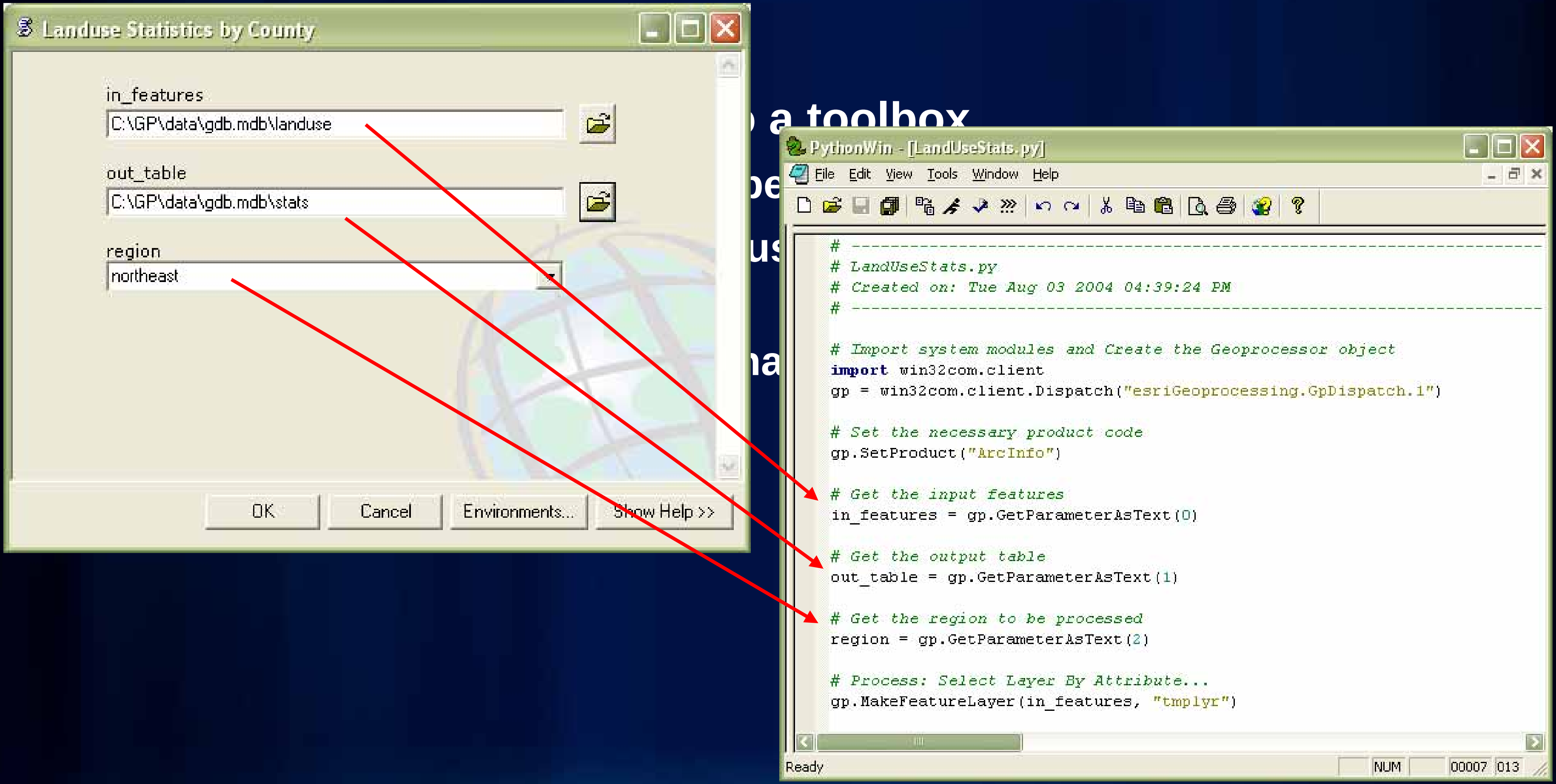
Creating Tools from Scripts

a toolbox

pe

us

na


Landuse Statistics by County
OK
Cancel
Environments...
Show Help >>
PythonWin - [LandUseStats.py]
File Edit View Tools Window Help
Ready
NUM
00007 013

in_features: C:\GP\data\gdb.mdb\landuse

out_table: C:\GP\data\gdb.mdb\stats

region: northeast

```
# ---------------------------------------------------------------------------
# LandUseStats.py
# Created on: Tue Aug 03 2004 04:39:24 PM
# ---------------------------------------------------------------------------

# Import system modules and Create the Geoprocessor object
import win32com.client
gp = win32com.client.Dispatch("esriGeoprocessing.GpDispatch.1")

# Set the necessary product code
gp.SetProduct("ArcInfo")

# Get the input features
in_features = gp.GetParameterAsText(0)

# Get the output table
out_table = gp.GetParameterAsText(1)

# Get the region to be processed
region = gp.GetParameterAsText(2)

# Process: Select Layer By Attribute...
gp.MakeFeatureLayer(in_features, "tmplyr")
```

Getting Input Parameter Values

- If a script is the source of a script tool, it can use the *GetParameterAsText()* function to access the input parameter values.

```
import arcpy

# Get the input feature class or layer
in_features = arcpy.GetParameterAsText(0)

# Get the input Field
in_fieldName = gp.GetParameterAsText(1)
```

Setting Output Messages

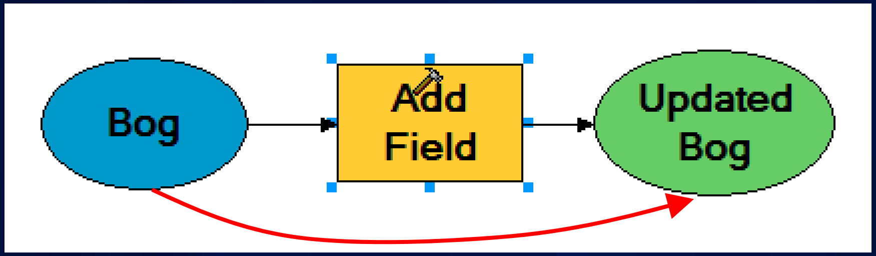
- When a script tool is executed, messages often need to be returned to the user, especially when problems arise

- ArcPy has several functions for adding messages:
  - AddMessage(string)
  - AddWarning(string)
  - AddError(string)

- Messages added to the ArcPy are immediately returned to the application or script executing the tool

Script Tools - Output Parameters

- All tools should have an output
  - If the script updates an input dataset, create a derived parameter
  - Set its dependency to the input parameter
  - The properties of the input are automatically added to the output



Value

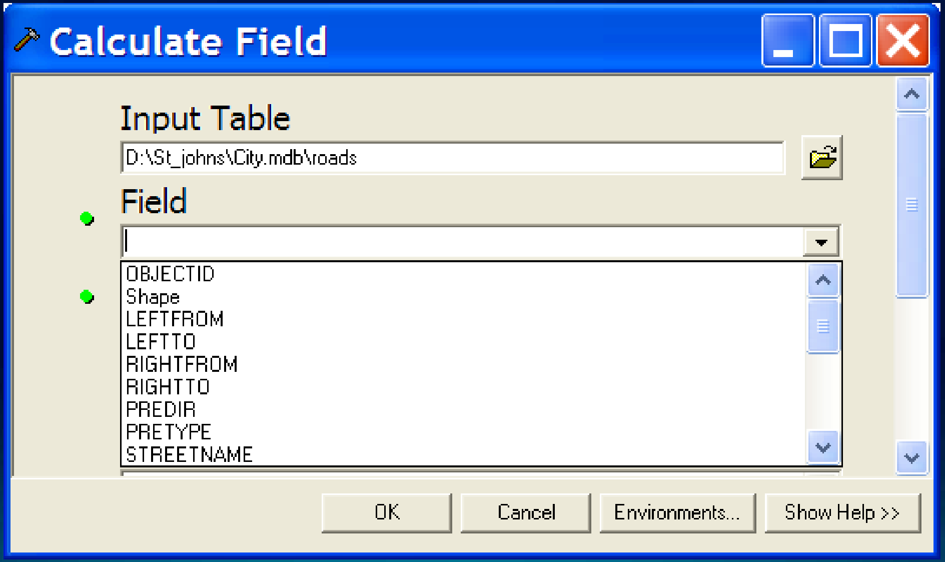
  - This makes for a better user experience when used in ModelBuilder

Script Tools - Output Parameters

- If an output parameter is a scalar value, make it derived
  - Use SetParameterAsText() function to set it at the end of your script
  - Allows chaining of the output value in a model
  - The output value is automatically added as a message

Script Tools - Parameter Dependencies

- Some parameter types have built-in behavior when there is a parameter dependency
  - Fields with an input table or feature class
    - Fields will be populated automatically in the dialog
  - Derived parameter with an input parameter
    - The derived parameter value will automatically be set to the value of the input parameter it depends upon


Calculate Field

Input Table
D:\St_johns\City.mdb\roads

Field

OBJECTID
Shape
LEFTFROM
LEFTTO
RIGHTFROM
RIGHTTO
PREDIR
PRETYPE
STREETNAME

Spatial Analyst module

- Integrates Map Algebra into Python
  - *Defines geographic analysis as algebraic expressions*

  - Includes all Spatial Analyst tools
  - Supports operators in Map Algebra expressions
  - Helper classes that can be used to support complex parameter
  - Output on the left-side

```
from arcpy.sa import *
demm = Raster("DEM") / 3.28
slpdeg = Slope(demm, "DEGREE")

demfs = FocalStatistics("DEM", NbrRectangle(3,3), "MEAN")
```

Raster class

- Returned output from Spatial Analyst tools
  - Can be used as inputs to tools and Spatial Analyst Map Algebra expressions
- Supports operators (or arithmetic operations in Map Algebra expressions)
- Has properties and methods for analysis
  - raster.min
  - raster.max
  - raster.save()

Raster Integration

- NumPy is a 3rd party Python library for scientific computing
  - A powerful array object
  - Sophisticated analysis capabilities
- Raster objects can be converted to NumPy arrays for analysis
  - RasterToNumPyArray(), NumPyArrayToRaster()

```
inras = "ras100"

# convert raster to Numnpy array
rasArray = arcpy.RasterToNumPyArray(inras)

# ARRAY SLICING: get the total sum of every third value
#  from every third row of the  raster
sampArray = rasArray[::3,::3]
sum = numpy.sum(sampArray)
print sum
```

Demo

- SA module (Map Algebra)

Additional Python Sessions

- **Using Python to Glue it all Together**
  - Wed 1:00pm Primrose A
- **Python Scripting for Map Automation**
  - Wed 2:45pm Primrose A
- **Administering your Enterprise Geodatabase with Python**
  - Wed 4:00pm Demo Theater 1 - Oasis 1

Questions?

Python IDEs

- Review of IDEs:
  - http://blogs.esri.com/Dev/blogs/geoprocessing/archive/2010/09/14/Review-of-IDEs-for-Python.aspx