2011 Esri Developer Summit
Palm Springs, CA

Hitchhiker's Guide to Python and ArcGIS

David Wynne and Jason Pardy

esri

Hitchhiker’s Guide to Python and ArcGIS

Hitchhiker's Guide to Python and ArcGIS

Python was introduced with ArcGIS 9, and with each subsequent release, the Python framework has been extended, providing more capabilities and a richer, more Python-friendly experience. Python extends across ArcGIS and becomes the language for data analysis, conversion, and management as well as map automation, helping increase productivity.

This session will start by introducing the capabilities of and advantages in using Python for working with ArcGIS. From there, the session will highlight constructing workflows in Python, working within the Python window (a fully integrated Python prompt within ArcGIS), building geoprocessing tools in Python, and maximizing modules for map automation and map algebra.

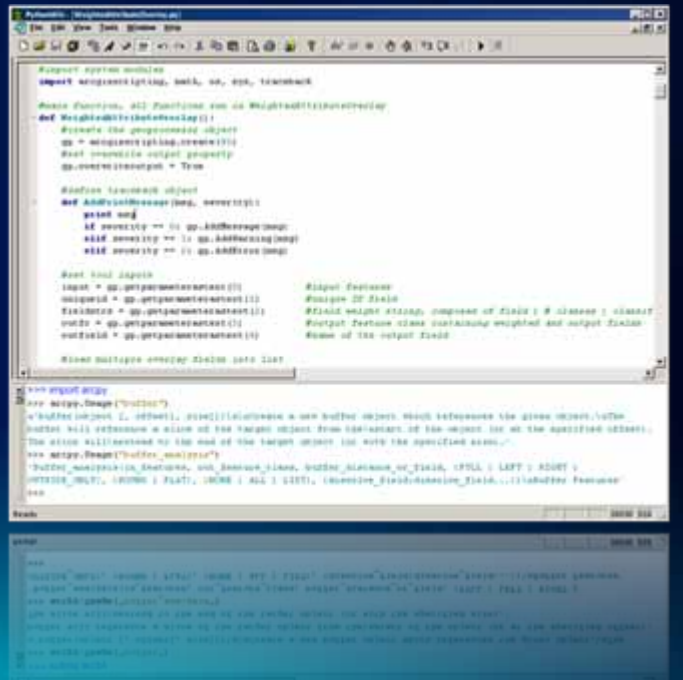
Prerequisite: None

Outline

- Essentials
  - Why use Python scripting?
  - Python 101
  - What's new in ArcGIS 10?
  - Executing tools
  - Messages and error handling
  - Cursors
- Automation
  - ArcPy functions
  - Batch processing
  - Map Automation
- Break (20 mins)
- Script tools
  - Tool design & validation
- Raster analysis and Map Algebra
- Calculating Fields with Python

Why Python?

- Python was designed to be easy to read and learn
- Easy to maintain
- Excellent for beginners and experts
- Suitable for large projects or small scripts
- Established and active user community
- Large collection of modules
- Cross platform



Python 101

- **Where do I write Python code?**
  - **Python window in ArcGIS**
  - **An an IDE like PythonWin, Wing, etc.**
  - **Review of IDEs:**
    - **http://blogs.esri.com/Dev/blogs/geoprocessing/archive/2010/09/14/Review-of-IDEs-for-Python.aspx**

Python 101

- Python has logic for testing conditions
  - if, else statements
  - Colon (:) at end of each condition
  - Indentation determines what is executed
  - == test equality; other operators like >, <, !=

```
var = 'a'
if var == 'a':
    # Execute indented lines
    print 'variable is a'
else:
    print 'variable is not a'
```

Python 101

- Techniques for iterating or looping
  - while loops, counted loops, list loops
  - Colon (:) at end of statement
  - Indentation determines what is executed

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

for num in range(1,5):
    print num

x = [1, 2, 3, 4]
for num in x:
    print num
```

Python 101

- **Function & Modules**
  - ***Function*: a defined piece of functionality that performs a specific task**
  - ***Module*: a python file (typically where functions live)**

```
import math
math.sqrt(100)
```

Python 101 – Python types

- **Take advantage of key Python types**

Type	Explanation	Example
Lists	Flexible ordered collection	L = ["10 feet", "20 feet", "50 feet"]
Tuples	An immutable list (not editable)	T = ("Thurston", "Pierce", "King")
Dictionaries	Key/value pairs	D = {"ProductName": "desktop", "InstallDir": c:\\ArcGIS\\Desktop10.0"}

Python 101 – Functions

- **Python functions are a simple way to organize and re-use functionality**

```
import arcpy

def increaseExtent(extent, factor):
    """Increases the extent by the given factor"""
    XMin = extent.XMin - (factor * extent.XMin)
    YMin = extent.YMin - (factor * extent.YMin)
    XMax = extent.XMax + (factor * extent.XMax)
    YMax = extent.YMax + (factor * extent.YMax)

    return arcpy.Extent(XMin, YMin, XMax, YMax)

oldExtent = arcpy.Describe("boundary").extent
newExtent = increaseExtent(oldExtent, .1)
```

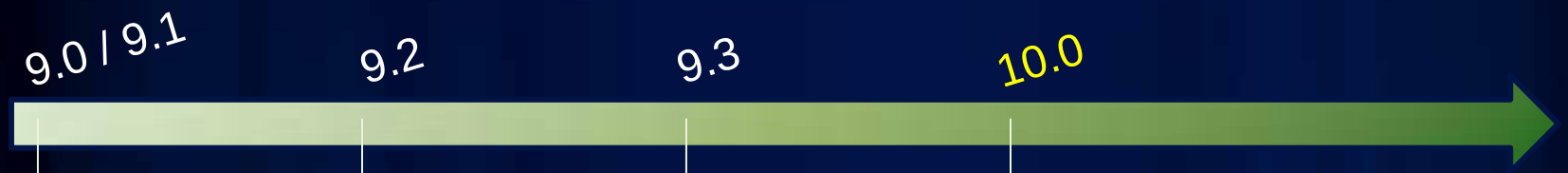
Demo

- Python Types & Functions

Scripting Fundamentals

- Provide an efficient method for defining and executing a workflow
- Create generic scripts that can be used multiple times
- Create new analytical tools

A brief history of Python in ArcGIS



- dispatch–based Geoprocessor
- *Python 2.1*

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

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

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

ArcPy

What is ArcPy?

- A cornerstone for automation in ArcGIS
  - Analysis, Conversion, Management, Map automation
- ArcPy is a native Python site-package
  - Access to 800+ geoprocessing tools
  - Functions, classes and modules
    - With embedded reference documentation
  - Code completion for ArcGIS components in your favorite Python editor
  - Builds on the arcgisscripting module *(pre-10)*

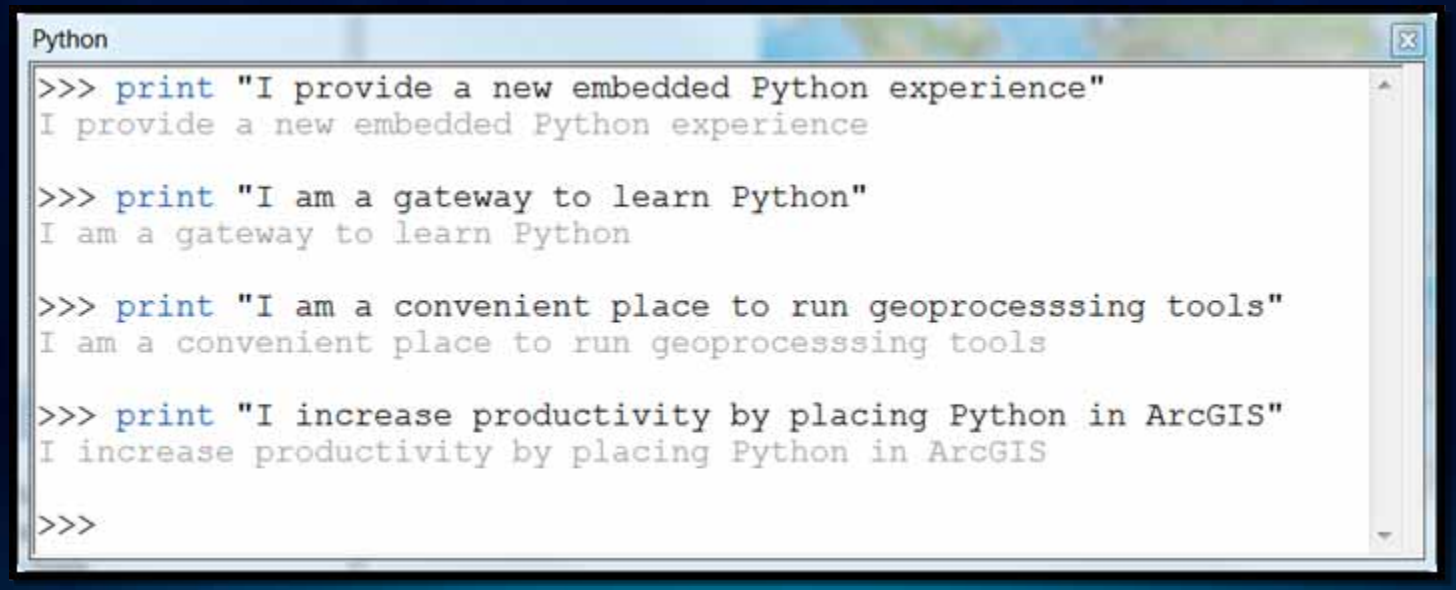
ArcPy

- **Improved coding experience, such as:**
  - **Cursors**
  - **Classes**
  - **Multi-value parameters can be expressed as Python lists**
  - **Ability to convert rasters to and from NumPy arrays**

- **ArcPy is supported by modules, including:**
  - **A mapping module (*arcpy.mapping*)**
  - **A Spatial Analyst module (*arcpy.sa*) to support map algebra**
  - **A Geostatistical Analyst module (*arcpy.ga*)**

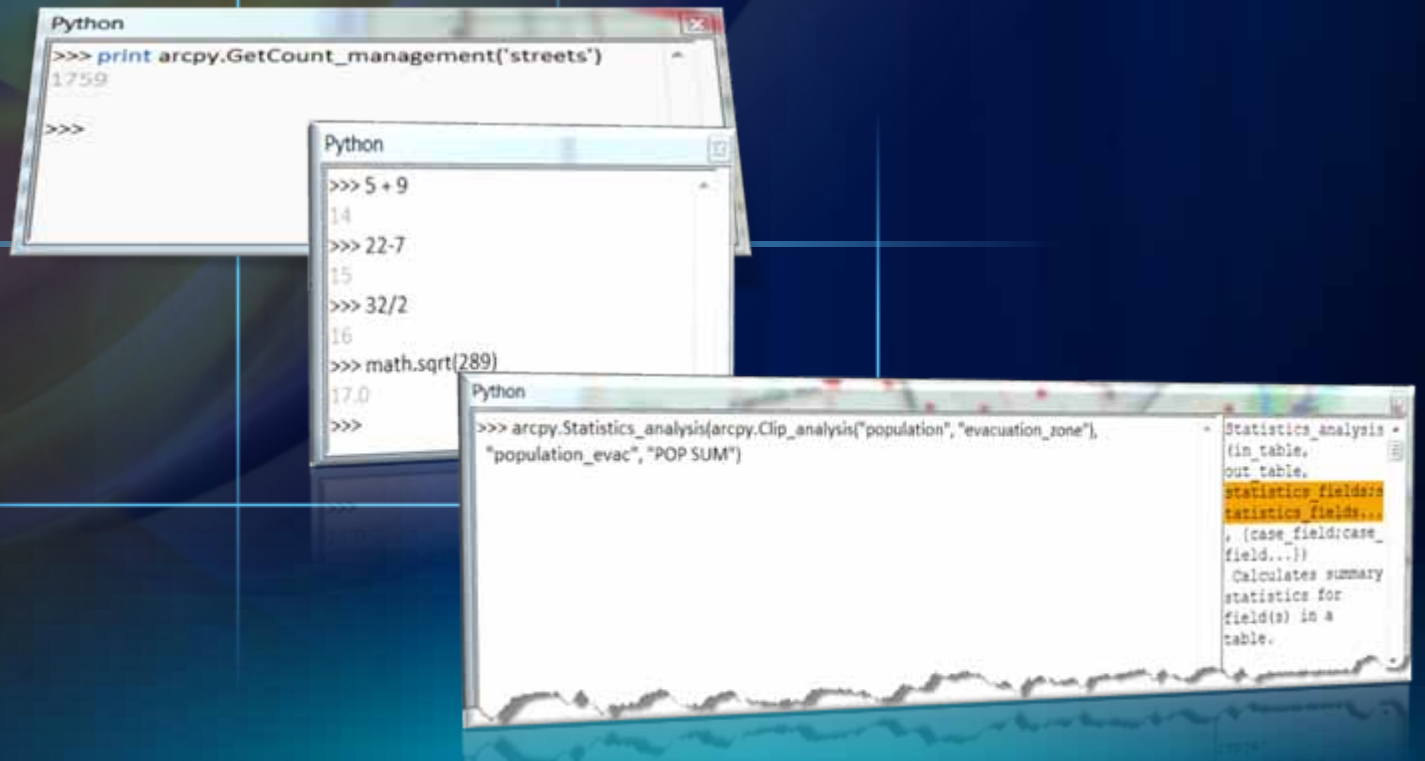
What is the Python window?

- **An interactive Python runtime embedded in ArcGIS**
  - **Can access ArcPy, including tools**
  - **Can access any other Python functionality**
  - **Better code completion and intelligence**



Demo

- Editing workflow



Running Tools

- Tools are accessed as functions on arcpy
- Environments as properties from arcpy.env class

```
# ~~~ PYTHON CODE ~~~
import arcpy

# Set the workspace
arcpy.env.workspace = "c:/st_Johns/GISData.gdb"

# Execute Geoprocessing tool
arcpy.Intersect_analysis(["roads", "urban_area", "urban_roads"],  5, "join")
```

*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

Tool Messages

- Tools return 3 types of messages
  -  Informative messages (severity = 0)
  -  Warning messages (severity = 1)
  -  Error messages (severity = 2)

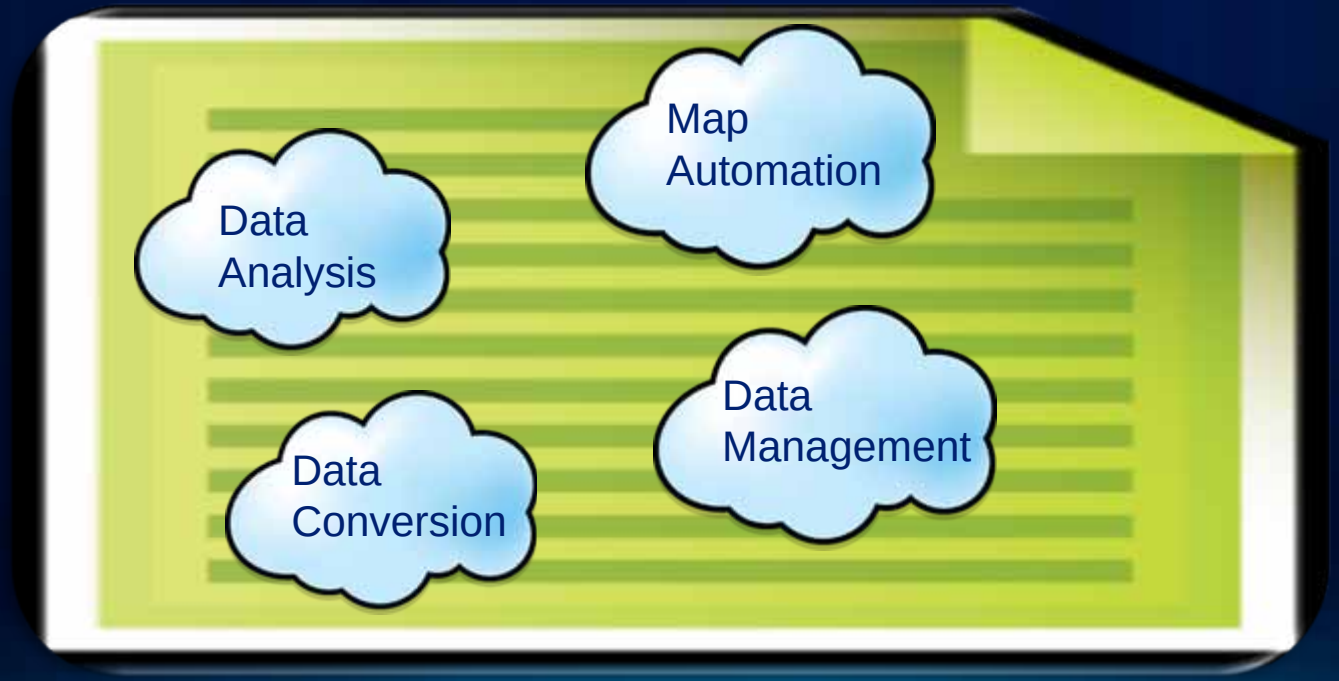
```
# start try block
try:
    arcpy.Buffer_analysis("c:/ws/roads.shp", "c:/outws/roads10.shp", 10)

# 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
```

Demo

- Setting environments
- Returning messages
- try…except statement

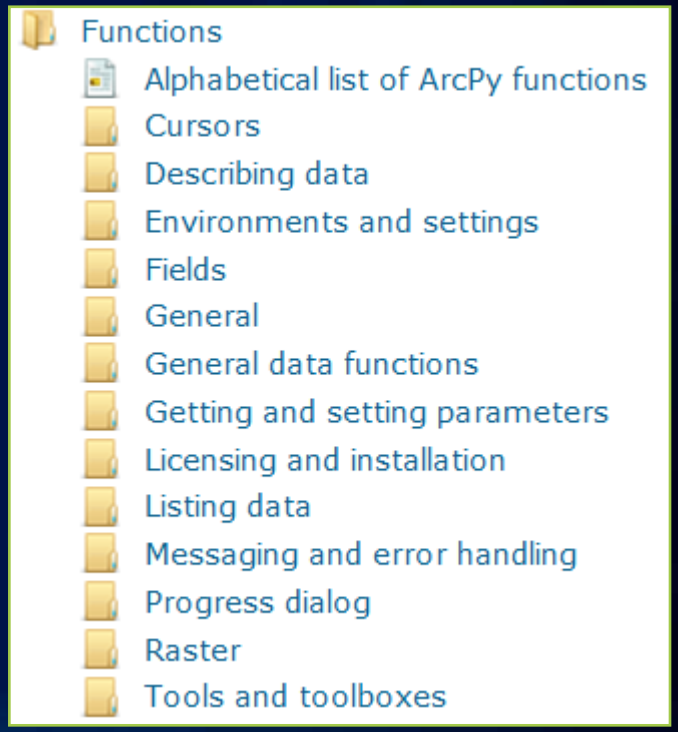
Automation = Productivity

- Python extends across ArcGIS to help automate common GIS tasks.



Functions

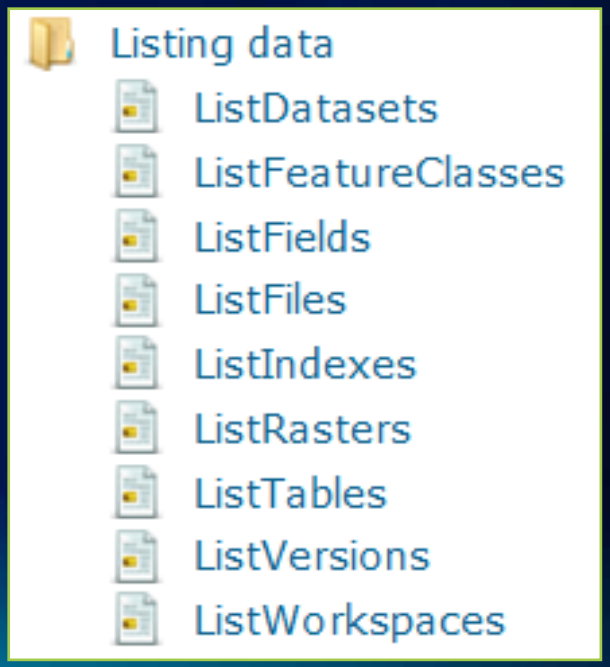
- **The ArcPy module contains functions necessary to perform many scripting tasks**
  - **Listing data**
  - **Describing data**
  - **Validating table and field names**
  - **Getting messages**
  - **etc.**
- **Allows automation of manual tasks**



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.**

- **ArcPy *List* functions exist to support these cases:**



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

- ArcPy supports a series of classes

- Classes are used to create objects
  - Once created, objects have methods and properties

- Classes are used most often used for:
  - Tool parameters
  - Working with geometry

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, ...
- Geometry objects have methods for relational operators

```
# 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

- Geometry has a hierarchy
  1. A feature class is made of features
  2. A feature is made of parts
  3. A part is made of points

- A None is used as a separator between rings (holes) in a polygon part

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 can 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
```

Demo

- Cursors

Writing out to a table

```
import arcpy
import random

cursor = arcpy.InsertCursor(table)

# Write a 100 new rows with random values
for x in range(1,100):
    row = cursor.newRow()
    row.setValue("RF", random.random())

    cursor.insertRow(row)

del cursor, row
```

Open an insert cursor

Create a new row

Assign a random value to the row

Insert the row into the cursor

Delete cursor and row objects to prevent locking of data

Break – 20 mins

Geometry as input

- Don’t necessarily have to go through the steps of creating a feature class, opening a cursor and adding features
- Can create geometry objects on the fly and use those directly as input to geoprocessing tools

```
# Create a list of point objects
ptList = [arcpy.Point(358331, 5273193),
          arcpy.Point(358337, 5272830)]

# Create an array from the list of points
lineArray = arcpy.Array(ptList)

# Create a polyline feature from the array
polyline = arcpy.Polyline(array)

# Copy out the geometry by adding as input to CopyFeatures
arcpy.CopyFeatures_management(polyline, "c:/base.gdb/newline")
```

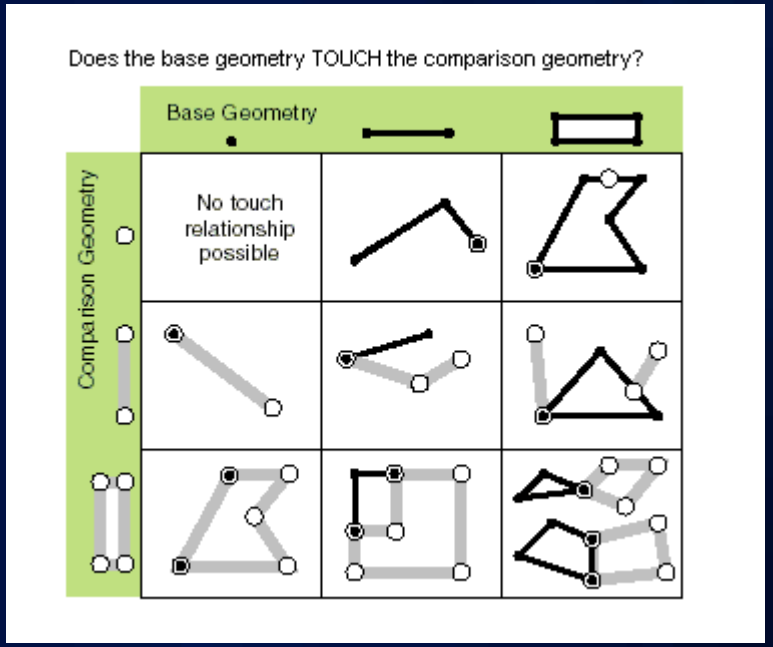
Geometry as tool output

- Can direct output to geometry by setting a Geometry object as the output parameter

```
arcpy.env.workspace = “c:/data/base.gdb”

# Returns a list of geometry objects
gList = arcpy.CopyFeatures_management(“rivers”, arcpy.Geometry())

# Print the extent of the first geometry in the list
print gList[0].extent
```

Geometry operators

- **Python geometry objects support relational operators at 10**
  - **contains**
  - **crosses**
  - **disjoint**
  - **equals**
  - **overlaps**
  - **touches**
  - **within**



Geometry operators

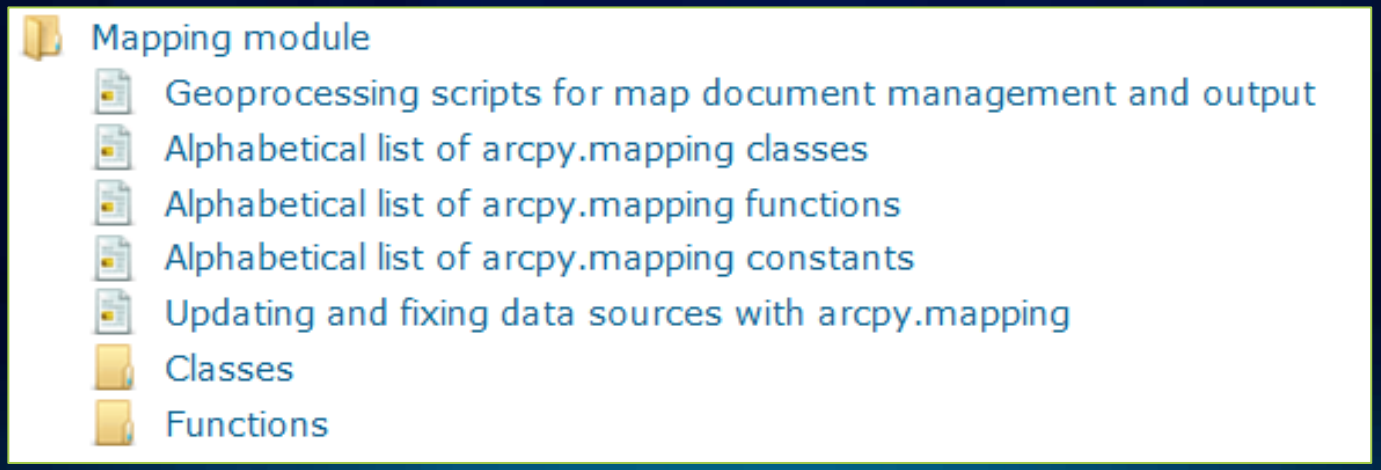
```
from arcpy import *
line1 = Polyline(Array([Point(1,10), Point(10,10)]))
line2 = Polyline(Array([Point(5,5), Point(7,15)]))

# Does line1 cross line2? crosses return a boolean
line1.crosses(line2)
```

Demo

- Geometry relational operators

arcpy.mapping module

- A python scripting API for:
  - Managing map documents
    - Repair/update data sources
    - Update a layer's symbology across many MXDs
    - Generate reports that lists document information
  - Exporting and printing of map documents
  - Map production/map series

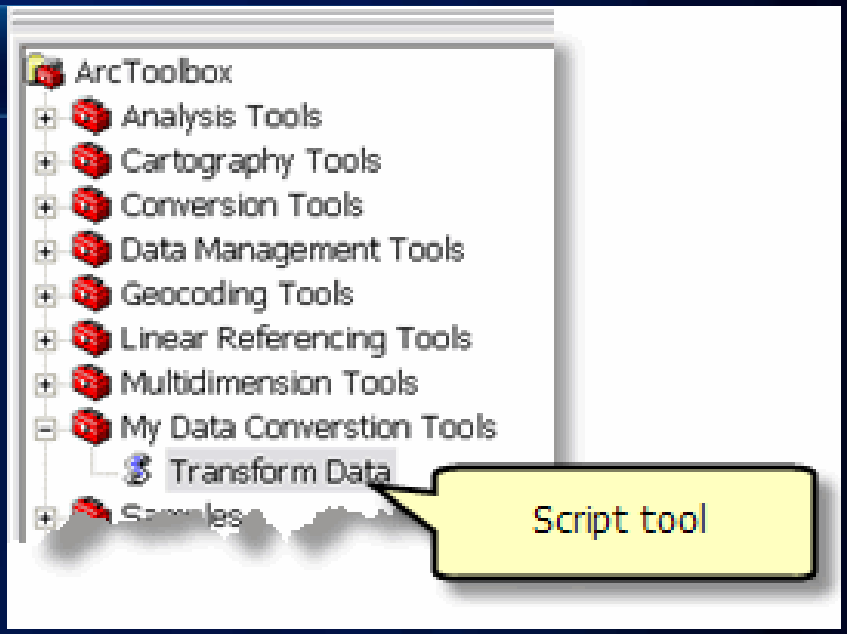


Demo

- Map Automation

Script Tools

- Script tools are the best way for creating and share custom geoprocessing functionality
- 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



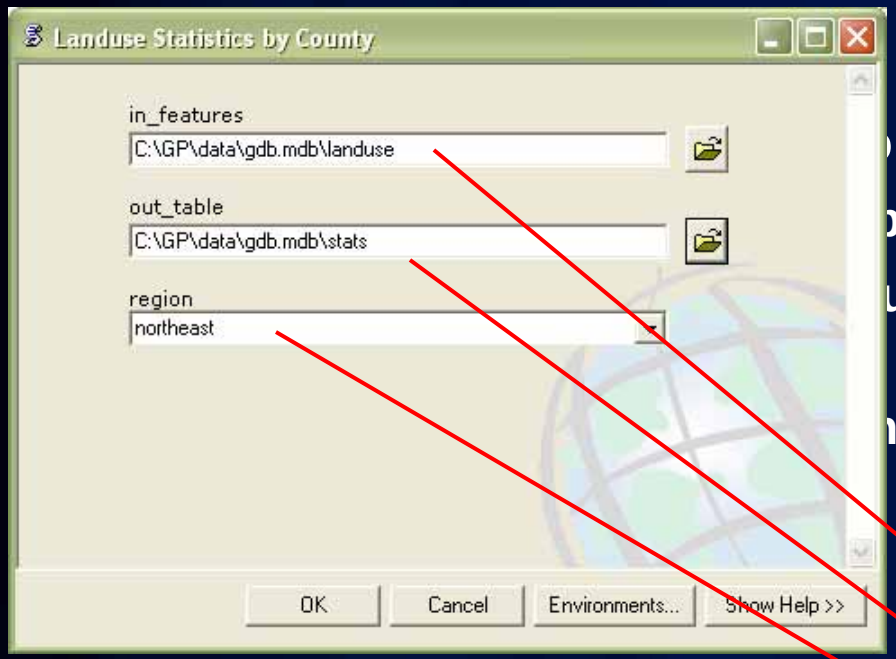
Creating Tools from Scripts

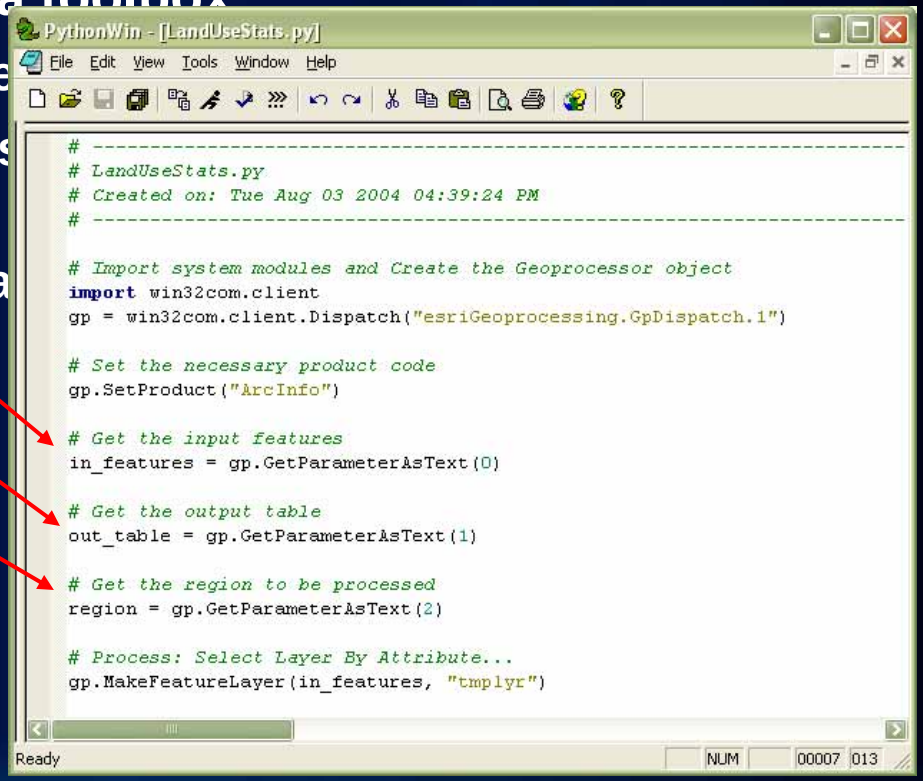
- Why?
  - The script is generic and can be used with other data
    - Script can use arguments from the user
  - You want to use a script in ModelBuilder
  - Easier to share your script
    - Not everyone knows how to run a stand-alone script
  - Puts a familiar face on your work

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





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.

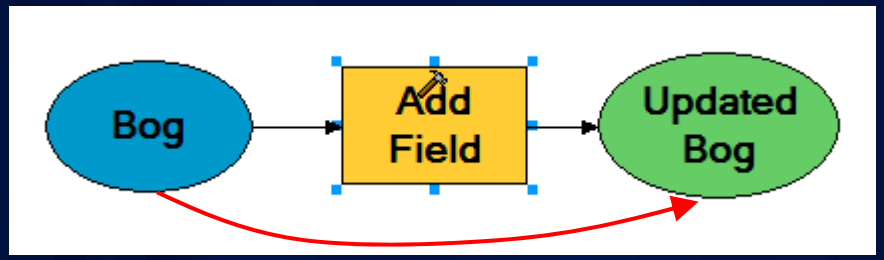
```
import arcpy

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

# Get the input Field
in_fieldName = arcpy.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 functions for adding messages:
  - AddMessage(“message”)
  - AddWarning(“message”)
  - AddError(“message”)
  - AddIDMessage(type, ID)
- 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

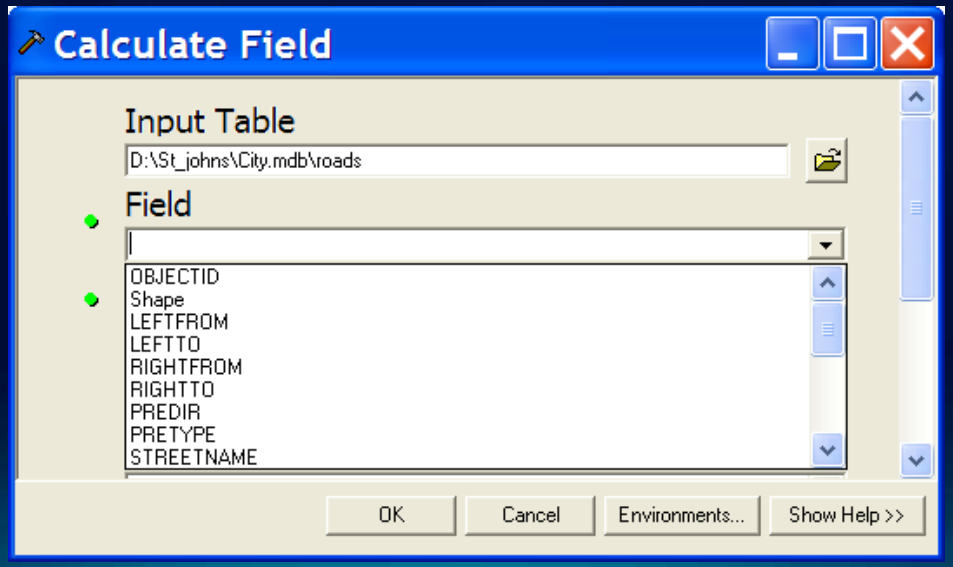
  - This makes for a better user experience when used in ModelBuilder

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**



Demo

- Script tools

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
  - Used to represent a raster dataset on disk
  - 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

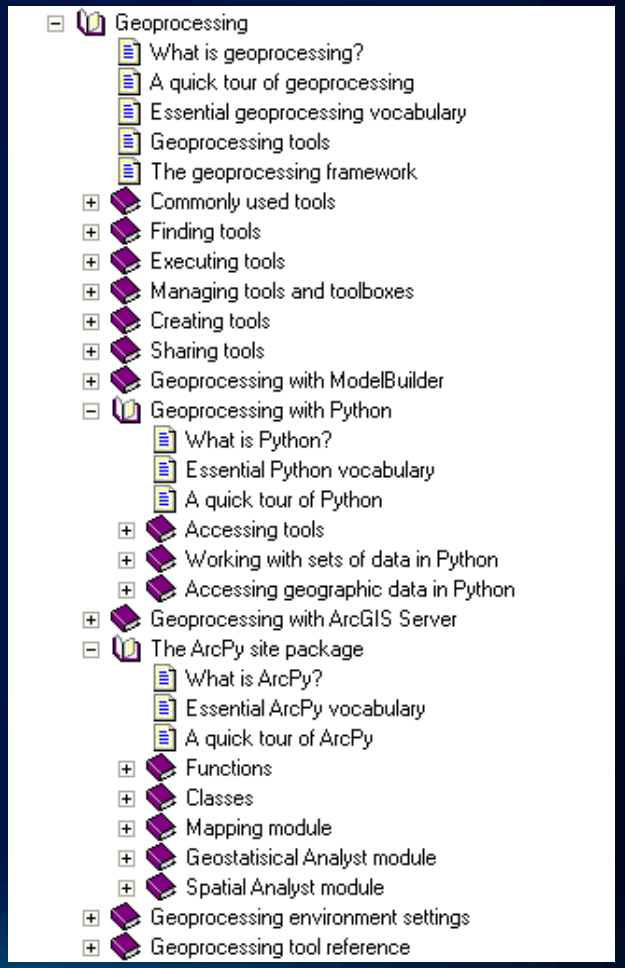
- SA module (Map Algebra)

TIP: Calculating Fields with Python

- Python provides easy solutions for performing field calculations using expressions & code blocks
- Use the help for examples and save time when you need to calculate fields…
  - Let's take a tour…
  - http://help.arcgis.com/en/arcgisdesktop/10.0/help/index.html#/Calculate_Field/00170000004m000000/

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**



Demo

- Geoprocessing Resource Center

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

Questions?