

# Advanced GIS Scripting with Python

John Porter

University of Virginia

# Why Program?

- With all the tools, wizards and programs available, there are still times when it pays to know how to do some simple programming
  - Tasks where looping or logical operations are important
  - Tasks where you want to automate a function to save having to point and click repeatedly!

# Learning to Program

The best way to learn to program, IS to program!

- Sources of information
  - Tutorials – good for getting started
  - Books - ditto
  - Language reference manuals – for getting the nitty gritty on how things work
  - Example code – never program what someone else has already done. Learn from them!

# Python

- Python is a language that is designed to be easy to write and read, so it's a good place to start!
- It also works quite well with the ArcObjects to provide a way to automate GIS tasks, but its not limited to GIS tasks alone

# Python Input and Output

```
print("Hello world")
```

Creates output:

```
Hello world
```

```
a=raw_input("What is your name? ")
```

```
print(a)
```

Prints out whatever you type in at the prompt

# Python Mathematics

```
a= 10
```

```
b= 20
```

```
c= 20.0
```

```
print ("a/b is: ")
```

```
print(a/b)
```

```
print ("but a/c is:")
```

```
print(a/c)
```

# Program Results

`a/b is:`

`0`

`but a/c is:`

`0.5`

Why are these different when b and c both are equal to 20?

b was 20 as an integer number

c was 20.0 as a real number

Rule: operations containing only integers yield integer results, but operations with one or more real numbers yield real number results!

# Hint

- To avoid odd problems in mathematical operations, use real numbers! 😊
- In all languages variables have “types” – in some languages they must be formally specified (e.g., C, Java)
  - Python tries to “guess” the appropriate type

# Common Variable Types

- Boolean (true or false – 1 or 0)
- Integer (e.g., 15)
- Floating point or Real (e.g., 15.0)
- String (e.g., “This is a test” or ‘This is a test’)

# Some more complex types for Python

- Tuples – lists of values that can't be edited or changed

```
mytuple=('a','b','c') # note ( )
print mytuple
('a', 'b', 'c')
print mytuple[1]
b
```

Anything after a # is a comment – not part of the program

Tuple elements are numbered starting with 0, so element [1] is b, not a

# Lists

- Lists – lists of values that can be edited or changed

```
mylist=['a','b','c\'] # note [ ]
```

```
print mylist
```

```
['a', 'b', 'c\']
```

```
print mylist[2] # numbers start at 0
```

```
c
```

# Built-in Functions

- Python has built-in functions that are always available. They take the form:

```
functionname(argument1,argument2..)
```

Sample:

```
a=raw_input("what is your name? ")
```

```
YournameLength= len(a)
```

len() is a function that gives the length of a string

# Type Conversion Functions

- `str(10)` – gives the string representation of 10 ('10')
- `int(5.5)` – gives the integer value (5)
- `float(5)` – gives the floating point value (5.0)

# Using functions from modules

- Functions are often stored in “modules” to use them you must first import the appropriate module

```
import math
```

- Functions have the format:
  - `module.functionname(arguments)`
- To get a square root

```
a= math.sqrt(10.0)
```

- [-] Main page
- [-] Global Module Index
- [+] What's New
- [+] Tutorial
- [-] Library Reference
  - [+] 1. Introduction
  - [-] 2. Built-In Objects
    - [+] 2.1 Built-in Functions
    - [+] 2.2 Non-essential Bu
    - [+] 2.3 Built-in Types
    - [+] 2.4 Built-in Exception
    - [+] 2.5 Built-in Constants
  - [+] 3. Python Runtime Service
  - [+] 4. String Services
  - [-] 5. Miscellaneous Service
    - [+] 5.1 pydoc -- Docume
    - [+] 5.2 doctest -- Test int
    - [+] 5.3 unittest -- Unit tes
    - [+] 5.4 test -- Regressior
    - [+] 5.5 test.test\_support
    - [+] 5.6 decimal -- Decima
    - [+] 5.7 math -- Mathema
    - [+] 5.8 cmath -- Mathem.
    - [+] 5.9 random -- Genera
    - [+] 5.10 whrandom -- Ps
    - [+] 5.11 bisect -- Array bi
    - [+] 5.12 collections -- Hig
    - [+] 5.13 heapq -- Heap c
    - [+] 5.14 array -- Efficient
    - [+] 5.15 sets -- Unordere
    - [+] 5.16 itertools -- Funct
    - [+] 5.17 ConfigParser -- (
    - [+] 5.18 fileinput -- Iterate

Note that `frexp()` and `modf()` have a different call/return pattern than their C equivalents: they take a single argument and return a pair of values, rather than returning their second return value through an 'output parameter' (there is no such thing in Py

For the `ceil()`, `floor()`, and `modf()` functions, the magnitude are exact integers. Python floats type (the platform C double type), in which case any fractional bits.

Power and logarithmic functions:

`exp(x)`  
Return  $e^{**x}$ .

`log(x[, base])`  
Return the logarithm of  $x$  to the given *base*. If the *base* is not specified, return the natural logarithm of  $x$  (that is, the logarithm to base  $e$ ). Changed in version 2.3: *base* argument added.

`log10(x)`  
Return the base-10 logarithm of  $x$ .

`pow(x, y)`  
Return  $x^{**y}$ .

`sqrt(x)`  
Return the square root of  $x$ .

Trigonometric functions:

`acos(x)`  
Return the arc cosine of  $x$ , in radians.

To learn about what functions are available and what the arguments are, you need to look them up in the documentation

# Documentation /Help

- Tutorial – sample programs and instructions (not comprehensive)
- Language Reference – Language details – variable types, expressions, statements – the rules!
- Library Reference – list of functions and common modules – the tools!

- [Main page](#)
- [Global Module Index](#)
- [What's New](#)
- [Tutorial](#)
- [Library Reference](#)
- [Language Reference](#)
  - [1. Introduction](#)
  - [2. Lexical analysis](#)
  - [3. Data model](#)
  - [4. Execution model](#)
  - [5. Expressions](#)
  - [6. Simple statements](#)
  - [7. Compound statements](#)
  - [8. Top-level components](#)
  - [A. History and License](#)
  - [Index](#)
- [Macintosh Reference](#)
- [Extending and Embedding](#)
- [Python/C API](#)
- [Documenting Python](#)
- [Installing Python Modules](#)
- [Distributing Python Modules](#)

# Python Documentation

Release 2.4.3  
29 March 2006

- [Tutorial](#)  
(start here)
- [What's New in Python](#)  
(changes since the last major release)
- [Global Module Index](#)  
(for quick access to all documentation)
- [Language Reference](#)  
(for language lawyers)
- [Library Reference](#)  
(keep this under your pillow)
- [Extending and Embedding](#)  
(tutorial for C/C++ programmers)
- [Macintosh Module Reference](#)  
(this too, if you use a Macintosh)
- [Python/C API](#)  
(reference for C/C++ programmers)
- [Installing Python Modules](#)  
(for administrators)
- [Documenting Python](#)  
(information for documentation authors)
- [Distributing Python Modules](#)  
(for developers and packagers)
- [Documentation Central](#)  
(for everyone)
- [Python How-To Guides](#)  
(special topics)

# What to Expect when you Start to Program?

- The computer will only do what you tell it to do, not what you want it to do!
- The learning curve is steepest at the beginning
  - As you get familiar with the jargon things get easier to understand
- “Playing around” – experimentation is key to developing understanding
- Writing a program takes 80% of the time. Testing and debugging take the other 80%.
- Programming can be FUN! 😊

# Program Logic

- The power of programming lies in the ability to write programs that make decisions and do repetitive tasks
- Loops
  - **WHILE**
  - **FOR**
- Decisions
  - **IF-THEN-ELSE**
  - **TRY** (other languages=Case, Switch)

# Sample Loop

```
x=5
```

```
while (x > 0): # note the :
```

```
    print("x =",x) # note the indent!
```

```
    x=x-1
```

```
Print "Done"
```

## ■ Output

```
('x =', 5)
```

```
('x =', 4)
```

```
('x =', 3)
```

```
('x =', 2)
```

```
('x =', 1)
```

```
Done
```

# Logical Operators

- $>$  - greater than
- $<$  - less than
- $==$  - equals (note double =)
- $<>$  or  $!=$  - Not equals
- $<=$  - less than or equal to
- $>=$  - greater than or equal to

# Using Character Strings

```
name = " "
```

```
while name <> 'John':
```

```
    name=raw_input("what is your name? ")
```

```
print("At last! Someone named John")
```

- Obviously <, >, <=, >= don't work with string data!

# Using IF

```
a=2
b=3
print("a=",a," b=",b)
answer=raw_input("What is a times b? ")
correctanswer= a*b
if (int(answer) == correctanswer):
    print("You are correct!")
    print("Very Smart!")
else:
    print("You are Wrong!")
print("done")
```

# Your program

- Alter the previous program so that it keeps asking you until you give the right answer!

# Surviving Programming

- Don't try to do everything at once
  - Write, test, write, test.....
- Each time you write a program try to add one new thing
- Patience is critical to survival
  - Twinkies and Caffeine can help!
- If you get stuck, “comment out” code you think might be the problem, then add it back in – step by step

# Using Python with ArcGIS

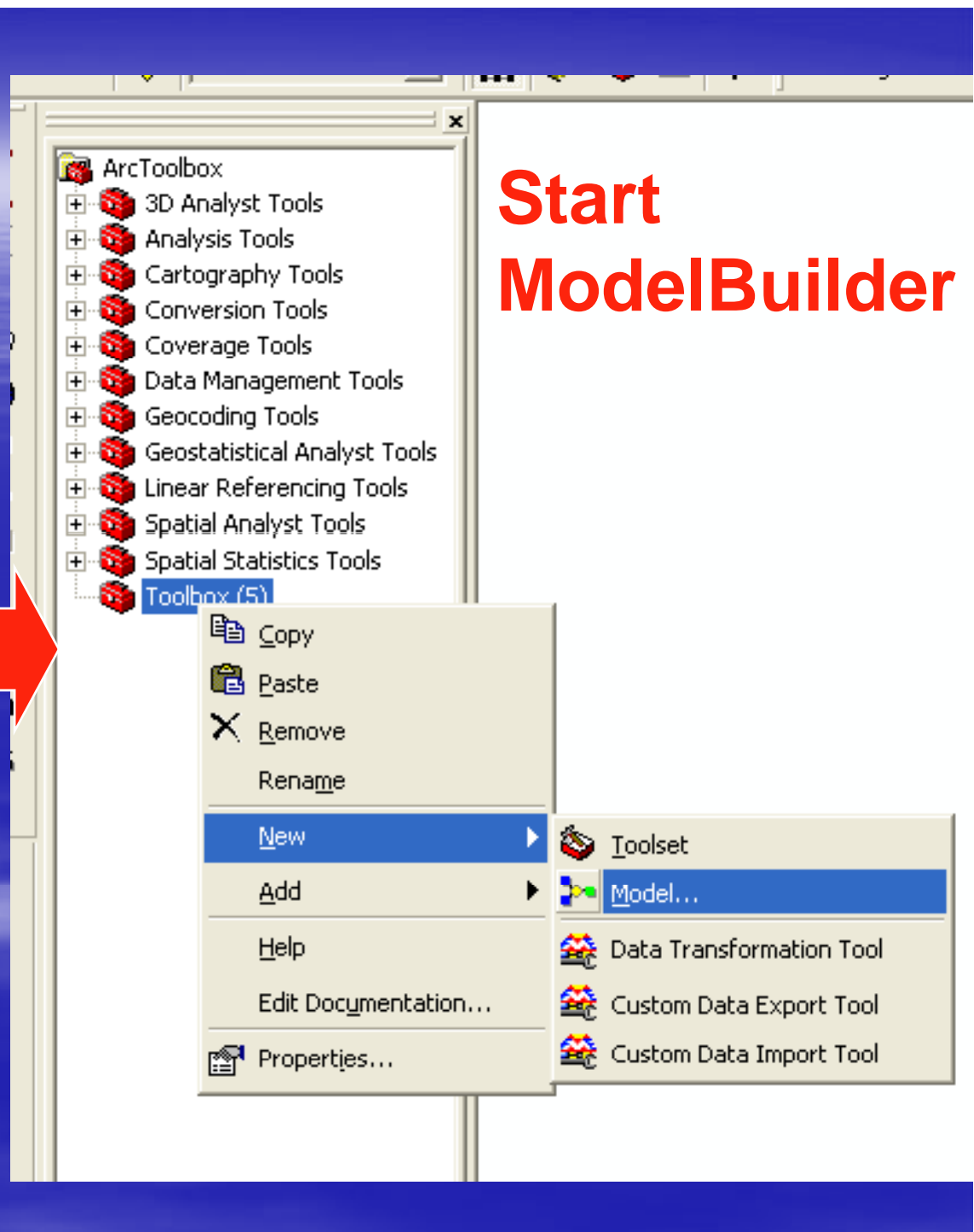
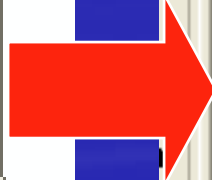
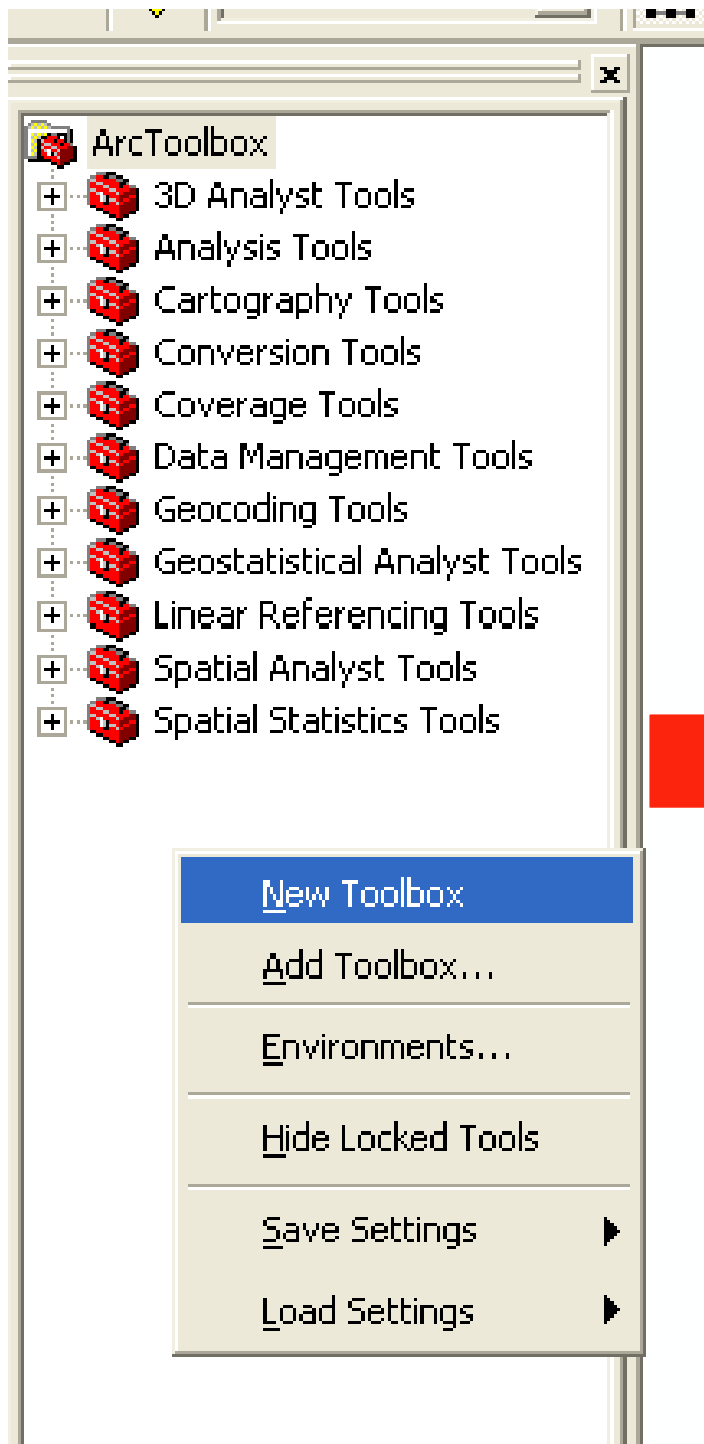
- Python can be used to create “scripts” that can do anything you can do with the tools in ArcGIS

# Why Script?

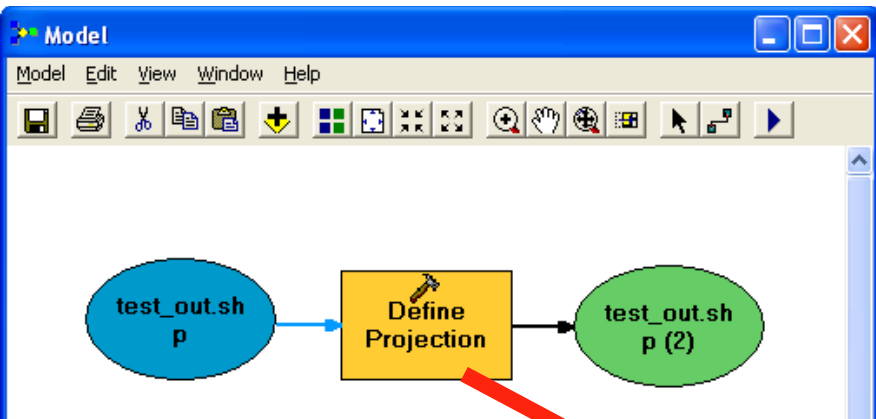
- The major reasons for using scripts are:
  - Looping
    - Repetitive tasks on a list of layers
  - Need for a simplified interface for user interactions
    - Prompts for selected parameters
  - The need for a command-line tool that can be run on a schedule without human intervention
    - Run weekly update on derived data layers

# Writing Scripts – The Easy Way

- Modelbuilder can be used to generate basic scripts that can be customized for more general uses



**Start  
ModelBuilder**



Set properties for projection

The screenshot shows the ArcMap interface with the 'Define Projection' dialog box open. The 'Input Dataset or Feature Class' is set to 'test\_out.shp' and the 'Coordinate System' is 'NAD\_1983\_StatePlane\_Virginia\_North\_FIPS\_4501'. A 'Spatial Reference Properties' dialog is also open, showing details for the selected coordinate system:

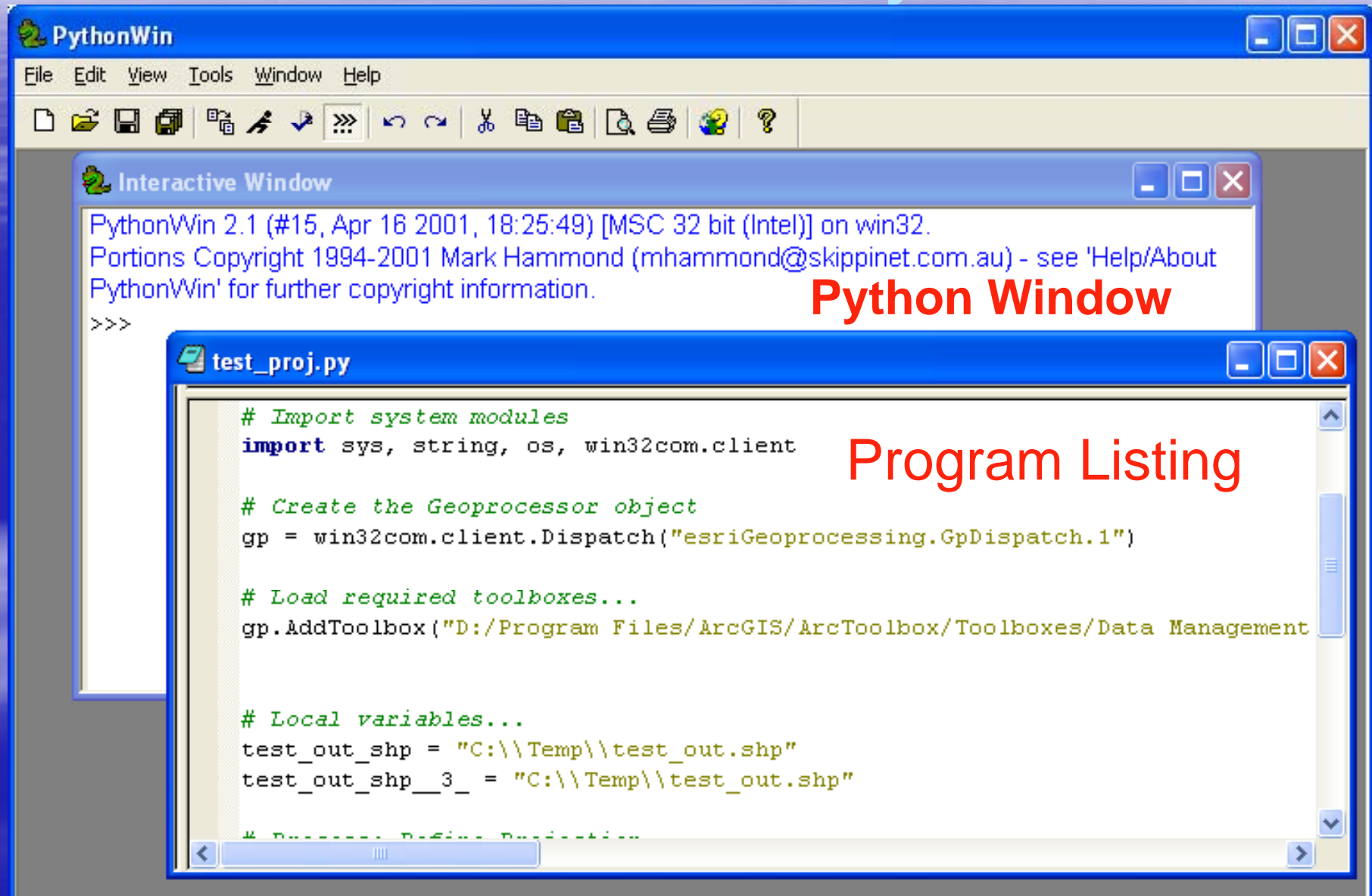
- Name: NAD\_1983\_StatePlane\_Virginia\_North\_FIPS\_4501
- Details:
  - Alias:
  - Abbreviation:
  - Remarks:
  - Projection: Lambert\_Conformal\_Conic
  - Parameters:
    - False\_Easting: 3500000.000000
    - False\_Northing: 2000000.000000
    - Central\_Meridian: -78.500000
    - Standard\_Parallel\_1: 38.033333
    - Standard\_Parallel\_2: 39.200000
    - Latitude\_Of\_Origin: 37.666667
    - Linear Unit: Meter (1.000000)
    - Geographic Coordinate System:

Buttons at the bottom of the 'Spatial Reference Properties' dialog include 'Select...', 'Import...', 'New...', 'Modify...', 'Clear', and 'Save As...'. The 'Define Projection' dialog has 'OK', 'Cancel', and 'Apply' buttons.

# Save Model to Script File

The image shows a screenshot of the 'Model' software interface. The main window displays a diagram with a yellow box labeled 'Define Projection' and a green oval labeled 'test\_out.sh p (3)'. A blue arrow points from the 'Export' menu item to the 'Define Projection' box. The 'Export' menu is open, showing options: 'To Graphic...', 'To Script', and 'Close'. The 'To Script' option is selected, and its sub-menu is open, showing 'Python...', 'JScript...', and 'VBScript...'. A red arrow points from the 'Python...' option to the 'Save As' dialog box. The 'Save As' dialog box is open, showing the file name 'test\_proj.py' and the save as type 'Python Scripts (\*.py)'. The dialog box also shows a list of folders in the 'My Downloads' directory, including 'aetna', 'ASM\_Dueser\_abstract\_files', 'BIRD\_FORMS', 'boxtran', 'CESM07\_taiwan\_files', 'cisco\_802', 'fday\_2005', 'fenster', 'LaSelva06', 'LC\_mtg', 'LinkSysWPS11', 'MODIS\_ASCII\_Sites', 'NEON\_0701', 'netgearwg11', 'NSFpanel', 'PHCK\_Dieoff', 'projection', and 'ShoreDEM13'.

# Edit the Model in PythonWin



The image shows a screenshot of the PythonWin application. The main window has a menu bar (File, Edit, View, Tools, Window, Help) and a toolbar with various icons. Two windows are open:

- Interactive Window:** Displays the PythonWin version (2.1), build number (#15), date (Apr 16 2001), and time (18:25:49). It also shows the operating system (MSC 32 bit (Intel)) and the platform (win32). Copyright information for Mark Hammond is provided. The prompt is `>>>`.
- test\_proj.py:** A code editor window showing the following Python code:

```
# Import system modules
import sys, string, os, win32com.client

# Create the Geoprocessor object
gp = win32com.client.Dispatch("esriGeoprocessing.GpDispatch.1")

# Load required toolboxes...
gp.AddToolbox("D:/Program Files/ArcGIS/ArcToolbox/Toolboxes/Data Management

# Local variables...
test_out_shp = "C:\\Temp\\test_out.shp"
test_out_shp_3_ = "C:\\Temp\\test_out.shp"

# Success: Define Prediction
```

**Python Window**

**Program Listing**

- Add the command:

```
gp.SetProduct("ArcInfo")
```

To get a license for the program to run even when ArcGIS is not.

# A Line-by-Line look at the Script

```
# -----  
# test_proj.py  
# Created on: Thu Mar 22 2007 02:07:29 PM  
#   (generated by ArcGIS/ModelBuilder)  
# -----  
# Import system modules  
import sys, string, os, win32com.client  
# Create the Geoprocessor object  
gp = win32com.client.Dispatch(  
    "esriGeoprocessing.GpDispatch.1")  
# Add this line to connect to the license  
# server when ArcGIS isn't running  
gp.SetProduct("ArcInfo")
```

# Attach to the Toolbox and Set the files to use

```
# Load required toolboxes...
gp.AddToolbox("D:/Program
Files/ArcGIS/ArcToolbox/Toolboxes/Data
Management Tools.tbx")

# Local variables...
test_out_shp = "C:\\Temp\\test_out.shp"
test_out_shp__3_ = "C:\\Temp\\test_out.shp"
```

# Do the Projection Definition

```
# Process: Define Projection...
gp.DefineProjection_management(test_out_shp, \
    "PROJCS['NAD_1983_StatePlane_Virginia_North_FIPS_4501', \
    GEOGCS['GCS_North_American_1983', DATUM['D_North_American_1
    983', \
    SPHEROID['GRS_1980',6378137.0,298.257222101]],\
    PRIMEM['Greenwich',0.0], \
    UNIT['Degree',0.0174532925199433]],\
    PROJECTION['Lambert_Conformal_Conic'],\
    PARAMETER['False_Easting',3500000.0],\
    PARAMETER['False_Northing',2000000.0],\
    PARAMETER['Central_Meridian',-78.5],\
    PARAMETER['Standard_Parallel_1',38.033333333333333],\
    PARAMETER['Standard_Parallel_2',39.2],\
    PARAMETER['Latitude_Of_Origin',37.666666666666666],\
    UNIT['Meter',1.0]]")
```


# Things to Note

- # defines the start of a comment. Anything following a # on a line will be ignored by the program
- Each command begins on a new line except:
  - When a “\” (backslash) is used at the end of a line, the next line is treated as a continuation of the previous line

# Things to Note

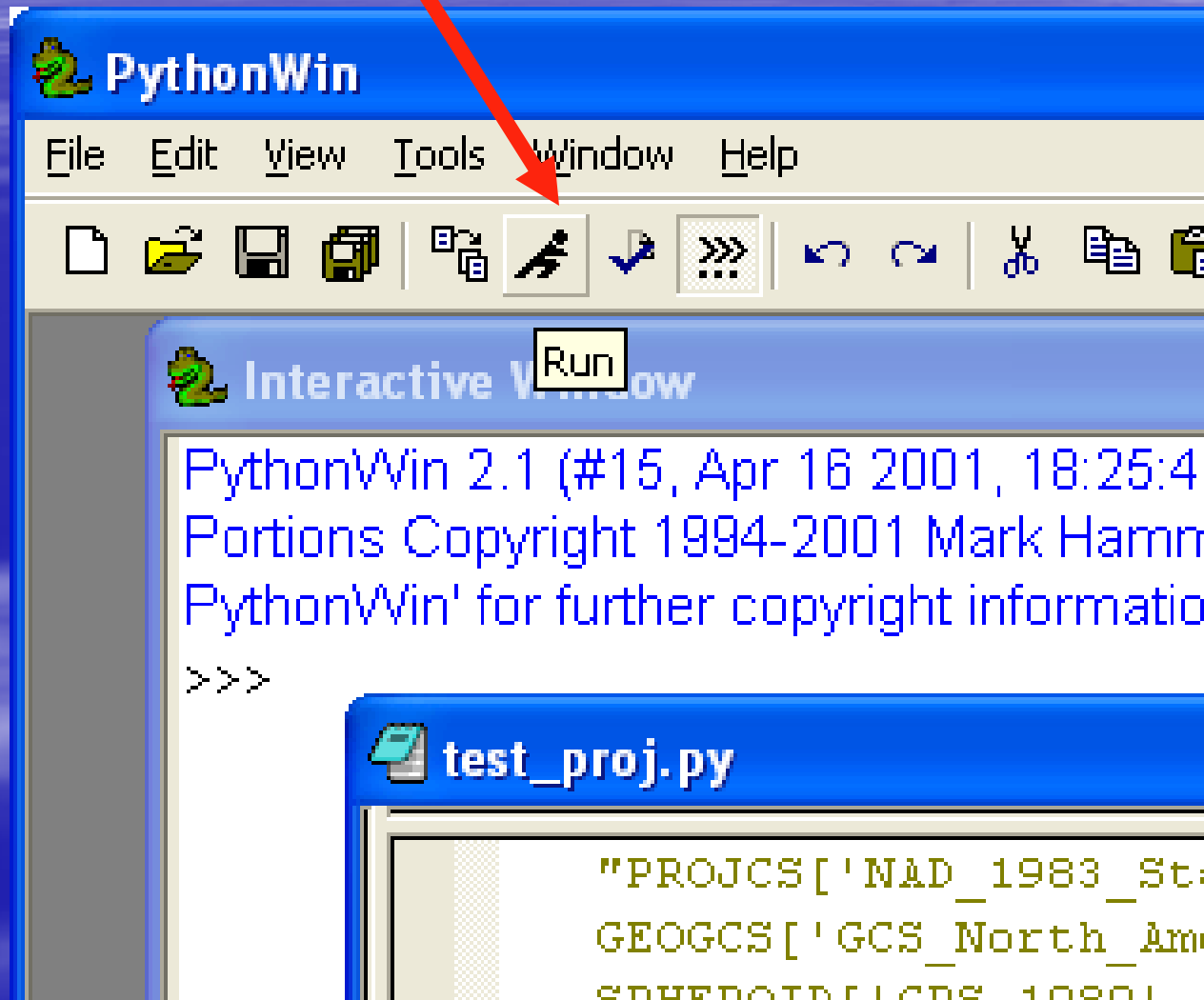
- In file paths, “/” not “\” is used – even on Windows computers that normally use “\” as separators
  - If “\” is used, it must be doubled “\\”
- Names of variables are NOT in quotes, but string values are. E.g.:

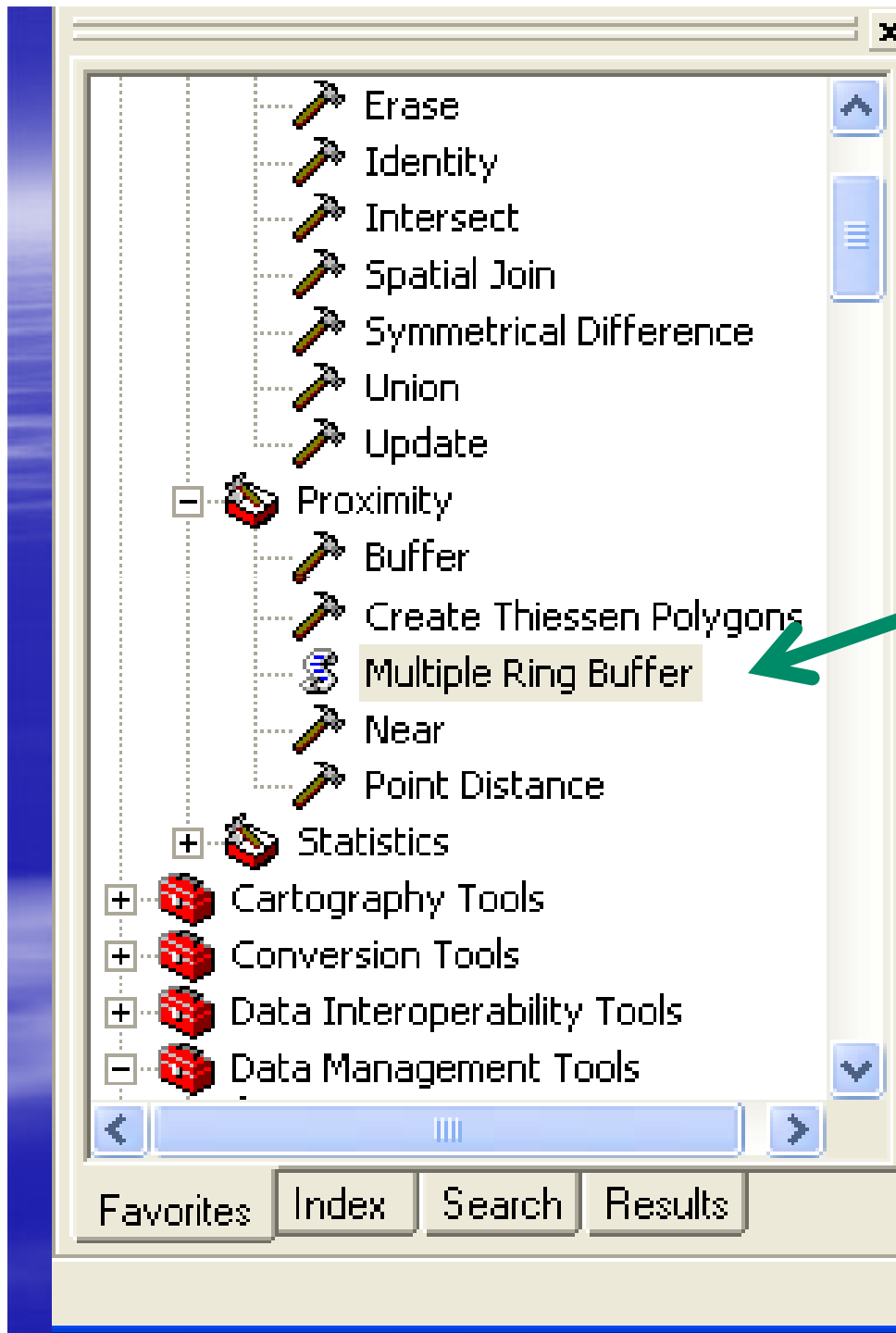
```
test_out_shp = "C:\\Temp\\test_out.shp"
```

 A variable named  
test\_out\_shp

 A string (or group of characters)  
Containing C:\\Temp\\ .....

# Click Here to “Run” the Script





Some tools in the ArcToolbox are python programs. You can open them for editing in the PYTHON interface to customize them for YOUR purpose. Be sure to save them with a new name, however!