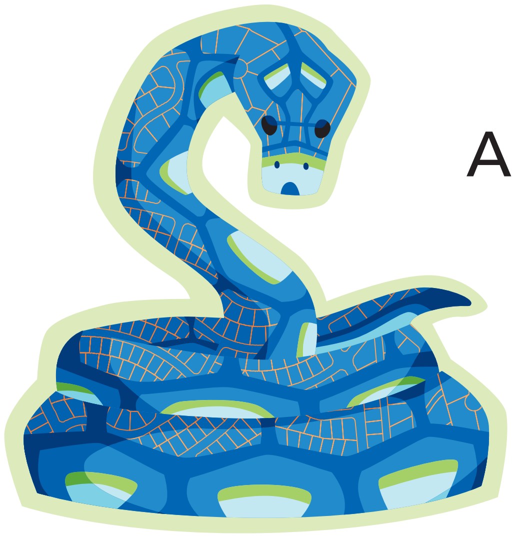

A Guide to the Python Universe for ESRI Users

By Howard Butler, Iowa State University

Scripting in ESRI software has historically followed two models. The first model is demonstrated by ARC Macro Language (AML). This model shows its PrimOS heritage. Output is piped to files, data handling is file system and directory based, and the code is very linear in nature.

The second model is exemplified by Avenue that shows its Smalltalk origins. Object.request is the name of the game: things don't have to be linear, I/O is sometimes a struggle, and integrating with other programs is a mixed bag. Both are custom languages that have their own dark, nasty corners.

With the introduction of ArcGIS 8, your scripting-based view of the world was turned upside down. Interface-based programming required you to use a "real" programming language, such as C++ or Visual Basic, to access the functionality of ArcGIS 8. There was no script for automating a series of tasks. Instead, you had to write executables, navigate a complex tree of interfaces and objects to find the required tools, and compile DLLs and type libraries to expose custom functionality.

With the introduction of ArcGIS 9, ESRI is again providing access to its software through scripting. ESRI realized that many of its users don't want or need to be programmers but would still like to have tools to solve problems they encounter. These tools include nice, consistent GUIs; scriptable objects; and the nuts-and-bolts programming tools necessary for customization.

To fulfill this need, ESRI supports a variety of scripting languages using ArcObjects—starting with the geoprocessing framework. Python, one of the languages supported, is an Open Source, interpreted, dynamically typed, object-oriented scripting language. Python is included with ArcGIS 9 and is installed along with the other components of a typical installation. This article gives you an overview of what is available in the Python universe to help you with GIS programming and integrating ESRI tools.

The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules,
Although practicality beats purity.
Errors should never pass silently,
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one—and preferably only one—obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea—let's do more of those!

Introducing Python

Python was first released in 1991 by Guido van Rossum at Centrum voor Wiskunde en Informatica (CWI) in the Netherlands. Yes, it is named after Monty Python's Flying Circus, which Guido loves. Its name also means that references from the movies and television show are sprinkled throughout examples, code, and comments. Many of Python's features have been cherry-picked from other languages such as ABC, Modula, LISP, and Haskel. Some of these features include advanced things, such as metaclasses, generators, and list comprehensions, but most programmers will only need Python's basic types such as the lists, dictionaries, and strings.

Although it is almost 13 years old, Python is currently at release 2.3. This reflects the design philosophy of the Benevolent Dictator for Life (Guido) and the group of programmers that continue to improve Python. They strive for incremental change and attempt to preserve backwards compatibility, but when necessary, they redesign areas seen in hindsight as mistakes.

The Design of Python

Python is designed to be an easy-to-use, easy-to-learn dynamic scripting language. What this means for the user is that there is no compiling (the language is interpreted and compiled on the fly), it is interactive (you can bring up the interpreter prompt much like a shell and begin coding right away), and it allows users to learn its many layers of implementation at their own pace.

The design philosophy of Python was most clearly described by Tim Peters, one of the lead developers of Python, in "The Zen of Python." Python programmers can use these maxims to help guide them through the language and help them write code that could be considered pythonic.

Python and GIS

Python provides many opportunities for integration within GIS computing systems. Cross-platform capabilities and ease of integration with other languages (C, C++, FORTRAN, and Java) mean that Python is most successful in gluing systems together. Because of the fluid language design, the development of large-scale applications is also easily supported. Many libraries and tools have already been developed for working with GIS data in Python. The basics are covered, including the manipulation of shapefiles, grids, and images, as well as more sophisticated stuff such as scripting ArcSDE and interaction with Web services and databases.

Vector Formats

A Python wrapper of the Open Source library Shapelib (shapelib.maptools.org) called pyshapelib is available for working with shapefiles. You can download it at www.hobu.biz/software/pyshapelib. It provides access to the individual vertices of the shape, access to the DBF file, and simple shape indexing. This library is useful if you want to manipulate the raw geometry of a shapefile or pan through the DBF file to get to specific records.

A Python wrapper of the Open Source library (gdal.maptools.org/ogr/) called OGR is available for working with vector formats other than shapefiles. These include MapInfo, ArcInfo coverage, PostGIS, Oracle Spatial, TIGER, SDTS, OPeNDAP, DGN, and Microstation DGN formats. OGR is part of the Geospatial Data Abstraction Library (GDAL), and it can be downloaded with the GDAL distribution (hobu.stat.iastate.edu/gdal-1.2.0win32.exe).

Using Python with Grid Data

A Python wrapper of the Open Source library GDAL (gdal.maptools.org) is available for working with ArcInfo grids. Many raster formats are supported by GDAL including JPEG 2000, BSP, United States Geological Survey digital elevation model, military elevation data, Enhanced Compressed Wavelet (ECW), Geographical Resources Analysis Support System (GRASS), TIFF/GeoTIFF, network Common Data Form (NetCDF), ERDAS IMAGINE, and Spatial Data Transfer Standard (SDTS). The Python library for Windows can be downloaded from hobu.stat.iastate.edu/gdal-1.2.0win32.exe. Many other formats, not listed here, are available.

GDAL, in combination with Numeric Python, gives you the flexibility to write map algebra operations using any format that suits your needs. For example, you could write a process that resided on a Web server, downloaded data with an Open Geospatial Consortium (OGC) Web Coverage Service, processed the data using some algebra, and delivered an image to the Web browser. The possibilities are endless once you have the ability to divorce data processing from data display.

Say you wanted to find the average value of a grid across both the rows and the columns. This is integer data in ArcInfo binary format. Using the interactive Python window, first import the GDAL library. Then tell GDAL where to find the ArcInfo coverage data file (.adf) for the grid that you want to open, using Python's raw mode to input the string. Pass the contents of the grid into a Numeric Python array and use Numeric's processing methods to produce the average. See Listing 1 for the code that performs these operations.

```
>>> import gdal
>>> gd =
gdal.gdal.Open(r'E:\gis\US_Elevation\usdem_2k\w001001.adf')
>>> . = gd.ReadAsArray()
>>> avg = Numeric.average(Numeric.ravel(array))
>>> avg
-0.0071967281963325313
```

Listing 1: Finding the average value of a grid across rows and columns

Projections

A Python wrapper from the Open Source library Proj.4 (proj.maptools.org) called py-Projection is available from Hobu GIS Consulting (hobu.biz/index_html/software/pyprojection/) for reprojecting data. Although it uses the European Petroleum Survey Group (EPSG) code system, you can define your own projections by using the raw parameters. Simply define the current projection, the x and y coordinates, and call a method that transforms them to the desired projection as shown in Listing 2.

Scripting ArcSDE

I developed a Python wrapper of the ESRI SDE C API called PySDE (hobu.stat.iastate.edu/pysde) that is available for writing scripts that manipulate ArcSDE. Almost all of the SDE C API is wrapped and has corresponding methods in Python that you can call. PySDE is Open Source, but you will need a licensed copy of the SDE C API to be able to use it. I developed PySDE because I felt there was a need to have the ability to prototype and script the ArcSDE engine. I wanted lean scripts

```
import Projection
albers = ["proj=aea",
"ellps=GRS80",
"datum=NAD83",
"lat_1=29.5",
"lat_2=45.5",
"lat_0=23.0",
"lon_0=-96.0",
"x_0=0.0",
"y_0=0.0"]
p2 = Projection.Projection(albers)
print '----------Albers--------------'
print 'Location: -93.00W, 42.00N'
print "Forward: ", p2.Forward(-93.00, 42.00)
print "Inverse: ", p2.Inverse(0.0, 0.0)
```

Listing 2: Using Python to project data.

Continued on page 36

A Guide to the Python Universe for ESRI Users

Continued from page 35

that ran on UNIX-like platforms without requiring ArcGIS to process the data. I have used PySDE to develop a specialized geometry algebra engine, administration scripts (drop this table, clean up log files, copy this data), and many data manipulation scripts. Another advantage of programming with PySDE is the immediacy of the Python interactive window. You can type in commands and see their effect in real time. This is a real time-saver when navigating complex hierarchies such as the SDE C API.

```
>>> from pyTS import TerraImage
>>> from pyTS import pyTerra
>>> apt = TerraImage.point(42.00, -93.00)
>>> drg = pyTerra.GetAreaFromPt(apt, ‘Topo’,’Scale64m’, 1,
1)
>>> doq = pyTerra.GetAreaFromPt(apt, ‘Photo’,’Scale64m’,
1, 1)
>>> drg.Center.TileMeta.Capture
‘1976-07-01T00:00:00.0000000-07:00’
>>> doq.Center.TileMeta.Capture
‘1994-04-18T00:00:00.0000000-07:00’
```

Listing 3: Getting the DOQ and the DRG map sheet dates from TerraServer.

Web GIS and Python

Python is perfectly suited for Web development. Web development with Python is often much faster than technologies such as Java or .NET. There are many tools available for doing Web development using Python such as Zope (www.zope.org), an application server; MapServer (mapserver.gis.umn.edu) a map-rendering server; and Twisted (twistedmatrix.com/products/download), a network protocol layer. Descriptions of other common Open Source tools for Web programming in Python with respect to GIS follow.

Web Services

Web services using Simple Object Access Protocol (SOAP), XML-remote procedure call (RPC), and Representational State Transfer (REST) clients are all the rage these days. Web services allow you to encode an XML-structured request to a server and have it respond back with XML-structured data. This architecture allows you to more easily separate the data storage and management portion of your system from the application side of the fence. Python provides many tools for working with Web services. XML-RPC is built right into the language, and many libraries are available for working with SOAP (pywebsvcs.sf.net) and REST.

pyTerra

One GIS Web service that is quite useful is the TerraService SOAP API. I have developed a Python wrapper called pyTerra (hobu.stat.iastate.edu/pyTerra) for easily interacting with the Microsoft TerraServer.

For example, if you would like to find the digital orthophoto quadrangle (DOQ) date of a specific longitude and latitude, one method would be to locate Federal Geographic Data Committee metadata, open it in a reader such as ArcCatalog, and record its value. While this method works, it is not scalable, and not practical if you have to look up the image dates for ten or fifteen thousand points. A program would be the only reasonable way to do it.

Fortunately, TerraServer stored the imagery acquisition dates along with the image data. You can easily use the Web services API that pyTerra provides to quickly access this information. The example in Listing 2 that gets the DOQ and the digital raster graphic (DRG) map sheet dates from TerraServer.

From this trace back of the Python Interactive Window, you can see that the DRG and DOQ dates are 1976 and 1994, respectively. These strings can be parsed into dates and inserted in a database, or the date information can be captured and used to burn a map image using the Python Imaging Library (PIL). Python makes it easy to work with Web services. Tools such as pyTerra can do much of the heavy lifting for you.

Conclusion

Python can provide you with a complete set of tools for your GIS toolbox. In combination with ArcGIS, the possibilities are endless. New technologies, such as Web services, are widely supported in Python. There are many online and paper resources to help you when developing Python scripts. A companion to this article (with links to sources) is available from my Web site at hobu.biz/software/python_guide_esri/.

Howard Butler
Center for Survey Statistics and Methodology
Iowa State University
Ames, Iowa 50010

Resources for Learning Python

Many available books give a general background to Python programming. Some of the best are from O'Reilly, but others from New Riders and Apress also supply a good introduction or cover specialized topics.

—*Howard Butler*

Learning Python, 2nd Edition
By Mark Lutz and David Ascher
At this point, *Learning Python* probably supplies the most complete book for an introduction to Python, especially for users coming from languages such as Visual Basic and Avenue. The examples and descriptions in this book are precise, relevant, and clear. I find myself going back to this book frequently, even though I have been writing Python since 1999. It teaches the basics well and shows you how to write pythonic code.
O'Reilly, 2003, 552 pp., ISBN: 0596002815

Python Essential Reference, 2nd Edition
By David M. Beazley
Python Essential Reference is truly a reference book. If you are already a proficient coder in another language, I would choose this book over *Learning Python.* It has everything you need and is very terse. A new edition should be in the works soon. This book mainly references Python 2.1—the version that ships with ArcGIS 9. Sams, 2001, 416 pp., ISBN: 0735710910

Programming Python, 2nd Edition
By Mark Lutz
Programming Python aspires to be the Python equivalent of the camel book in the Perl world (i.e., *Programming Perl,* also from O'Reilly). The updated second edition ballooned to more than 1,200 pages and comes with a CD–ROM. There are plenty of choice bits to devour. Of all the books listed, it supplies the best coverage of using the Python C API (which shouldn't go out of date too quickly now that it has been updated for Python 2.2). My criticisms of this expensive book would be that it is too verbose, is targeted too clearly at UNIX programmers, and uses larger example applications. Also, because it is so large, holding the book in your lap is problematic. I mainly use it as a Python C API reference and look to other books when I need help with specific things.
O'Reilly, 2001, 1,292 pp., ISBN: 0596000855

Dive Into Python
By Mark Pilgrim
Dive Into Python is an excellent book for those who already have some programming experience, and it approaches a wide range of topics at an intermediate level. This book provides good coverage of object-oriented programming in Python, unit testing, Web services, regular expressions, and performance testing. The entire text of this book is available online at www.diveintopython.org/. Apress, 2004, 4,113 pp., ISBN: 1590593561

Python in a Nutshell
By Alex Martelli
Python in a Nutshell is likely the most up-to-date, complete, and most poetic Python book available. I had the pleasure of eating lunch with the author, Alex Martelli, at the 10th Annual Python Conference. He is a very articulate speaker, and this carries over to his writing. This book covers the breadth of Python. Each chapter covers a separate problem domain and gives an overview (with great detail) of what is possible with Python. O'Reilly, 2003, 600 pp., ISBN: 0596001886

Python Cookbook
By Alex Martelli and David Ascher
The *Python Cookbook* was written by users of Python. A Web site, outlined in an O'Reilly Network article, was developed. Users submitted recipes that showed how they had solved problems with Python. The authors took the recipes, organized them into chapters, and made them coherent. There are some real gems here, especially in the algorithms chapter edited by Tim Peters. It also gives a good overview of how people express problems in Python, how they solve them, and tips that make life easier. The book conveys the community of Python as well as its problems. I recommend this book as a third or fourth Python book after you've covered the basics. O'Reilly, 2002, 606 pp., ISBN: 0596001673

Python Programming on Win32
By Mark Hammond and Andy Robinson
Those using the geoprocessing scripting engine in ArcGIS 9 will find *Python Programming on Win32* useful. It almost exclusively covers using the authors' Python COM extensions for Windows and the general usage of COM. It also describes how to script using the IDispatch interface exposed by Excel and how to work at the systems level with users, groups, and files. I also expect a second edition of this book in the near future because the COM extensions for Python have changed substantially in the years since this book was written. O'Reilly, 2000, 669 pp., ISBN: 1565926218

Jython Essentials
By Samuele Pedroni and Noel Rappin
Jython is a version of Python that runs on the Java Virtual Machine. It provides native access to Java classes and keeps the productivity of Python in a Java environment. This book covers the basics but also includes things such as using Jython in a JSP environment, which I found very handy for doing ArcIMS development. I was able to develop my IMS maps much quicker using Jython than equivalent straight JSP, and I recommend it if you find yourself in a similar situation. O'Reilly, 2002, 204 pp., ISBN: 0596002475

Online Resources

Documentation and online articles are probably the best way to stay abreast of updates to Python software, new techniques and methods, and new libraries that add capabilities to the language.

Resource	URL
Python Newbies Page	www.python.org/doc/Newbies.html
Python How-Tos	py-howto.sourceforge.net/
O'Reilly Python DevCenter	www.onlamp.com/python/
Daily Python	www.pythonware.com/daily/
Python Beginners' Mistakes	zephyrfalcon.org/labs/beginners_mistakes.html
Dive Into Python	diveintopython.org/
Thinking in Python	www.mindview.net/Books/TIPython
Data Structures and Algorithms with Object-Oriented Design Patterns in Python	www.brpreiss.com/books/opus7/