Example Source Code: http://www.paulmcnett.com/vfp/09MCNESC.zip
This article originally appeared in the September 2003 issue of FoxTalk.
If you enjoy this article, please consider sending a PayPal donation to p@ulmcnett.com. Thanks!

Exploring Python from a Visual FoxPro Perspective

Paul McNett

Visual FoxPro is quite capable in almost all aspects, except for application deployment – according to Microsoft, you may only deploy your applications to Windows systems. Feeling somewhat limited by this situation, I discovered Python, an open-source programming language that is a compelling choice for seasoned Visual FoxPro developers – and it runs on multiple platforms! In this article, I'll introduce you to Python, and show you how to access Visual FoxPro data with it.

Many Visual FoxPro developers feel that their choices of development environments are effectively limited to Visual FoxPro or .NET – Visual FoxPro because that is what they know, and .NET because it is compelling, capable, and rich with features and an astronomical advertising budget. Both of these options link against the Windows GUI, and are therefore limited to running on the Windows platform (unless you want to use a workaround such as Wine or a virtual machine such as Win4Lin or VMWare). This limitation is obviously by design. Other languages and development environments exist, separate from .NET, such as the various flavors of C, C++, and Java, that are multiplatform-capable, but in my quest for a multiplatform programming language, one rose to the top precisely because of its similarities in lots of ways to the experience of programming in FoxPro. I'd like to share what I've learned about Python, and how this knowledge may apply vis-a-vis Visual FoxPro.

You are probably asking "why should I care, why should I learn yet another language?" To that I must answer: just have a look, follow my tutorial and see what Python's strengths and weaknesses are. If it isn't for you, you haven't invested much to find that out. But if you end up appreciating it, as I know you will, you'll have another valuable tool to put in your kit.

Python at a Glance

Sporting a snake icon, Python was actually named after the "Monty Python's Flying Circus" BBC television show, by its creator, Guido van Rossum, starting in the late 1980's. It comes with an enthusiastic community of very smart people that are on the whole very happy to share their recipes and help newbies get started.

Python is an interpreted language with an elegant and readable syntax, and like Visual FoxPro you can mix-and-match OOP with functional programming, all variables are in effect variants, and there is a strong and flexible -but quite different- data model. Python consists of the core modules and is augmented by additional modules contributed by developers all over the world.

Python comes with a liberal open-source license, which means that you are free to modify and redistribute the code pretty much as you see fit. The Python interpreter is compiled C Code, and doesn't have any dependencies on any GUI, so therefore is quite portable to all known platforms, even including DOS, older versions of Windows, and embedded systems. Write your code once on a given platform, and as long as you haven't linked against any modules that are platform-dependent, your program will execute on any other platform. While most Python programs are built to run on Windows and Linux, Macintosh OSX is also widely supported and any Python program can be made to run on many other platforms as well.

Error handling is easy with the now-ubiquitous TRY...CATCH...FINALLY blocks which can be nested. There are standard exceptions that can be caught or thrown, and you can make your own as well by simply subclassing the Exception class. Unit testing and documentation are so natural and easy that you actually have to try to ignore these important features.

Python is strict-case: capitalization matters in your source code. However, depending on the operating system you are running it on, capitalization may or may not matter with regard to file and directory names.

Functions are first-class objects, which means that you can get an object reference to a function (either a standalone function or a method of a class), and pass it around to other functions. Garbage collection happens automatically in the background, and can be overridden if needed.

Source code is plain-text, which means you can use your favorite editor, and Python source files (scripts) by convention have a .py extension. Compilation to byte-code happens in the background by the Python interpreter as needed, which generates .pyc or .pyo files, depending on how you have it set. You may also, as in FoxPro, fire up the interactive Python interpreter, and execute statements line-by-line. The interactive interpreter is actually more capable than Visual FoxPro's command window, in that you can even execute control structures and define entire classes or functions.

Python, unlike Visual FoxPro, does not come with any graphical widgets for constructing Windows applications. It is, however, distributed with a multiplatform GUI toolkit called Tkinter, which you are free to use. In my opinion there are better choices than Tkinter (PyQt and wxPython to name the top two), but my point here is that Python programs are simple scripts that can optionally link to a GUI. It is nice not carrying along the GUI baggage when a given program doesn't need it, and it is very liberating to be able to choose for yourself the GUI that works best for you.

Python comes with a rudimentary Integrated Development Environment called IDLE, although I usually just use the command line because I find that simpler and more efficient for my purposes. I'll utilize IDLE in this tutorial since most Windows developers are more comfortable using a GUI than a command line.

Installing Python

Python is easily installed on Windows by downloading and running the installer at http://www.python.org/ftp/python/2.2.3/Python-2.2.3.exe. While the rest of this article assumes a Windows installation, you may install it on other platforms as well by following links at http://www.python.org/download. Make sure you leave all installation options checked so that you have everything you need. Please also install the win32all extensions (http://starship.python.net/crew/mhammond/downloads/win32all-152.exe), which includes functionality for Windows development. Python and the win32all extensions are completely free, as in freedom and as in beer.

After you install Python and the win32all extensions, I'll guide you through a short tutorial and show you how easy it is to talk to FoxPro data files. Along the way you'll learn Python's basic data types, control structures, and OOP implementation.

A Short Tutorial – Data Access in Python and Visual FoxPro

This tutorial will show how to use Python to manipulate tables created in Visual FoxPro. Along the way, you'll be introduced to Python's basic functionality. In the real world, I no longer use native Visual FoxPro tables, but rather choose to use MySQL (or other database servers) on the backend. From Visual FoxPro, I connect to MySQL using the MyODBC driver, and from Python, I connect using the MySQL client library directly. For simplicity in this article though, I'll just use native Visual FoxPro tables and then manipulate those tables both from VFP and Python.

The first thing you'll want to do is make sure your Python installation was successful. While there are many ways to use Python, I'll guide you through using the Python command line interpreter as implemented by IDLE, the Python IDE. Have a look in your Windows start menu, under Programs/Python 2.2. Click on the shortcut labeled “IDLE (Python GUI)”. Doing this will bring up a window titled “Python Shell”, complete with a command interpreter and a blinking cursor. The command prompt is “>>> ”. To aquaint you with the most basic Python commands, follow these examples:

```
>>> print “hello world!” # the string is output
hello world
>>> “hello world!” # a string object is output
'hello world'
>>> 1 + 2  # Addition
3
>>> 1 - 2  # Subtraction
-1
>>> 75 * 57 # Multiplication
4275
>>> 1 / 2 # Division of integer
0
>>> 1.0 / 2  # Division of float
0.5
>>> breakfast = “eggs”  # binding name to string
>>> count = 3 # binding another name to integer
```

```
>>> # simple output formatting/line continuation:
>>> print “for breakfast you'll have %d %s.” \
        % (count, breakfast)
for breakfast you'll have 3 eggs.
>>> import sys   # importing external module
>>> sys.platform
'win32'
>>> import odbc  # test if win32all is installed
>>> def myFunction(string):
        print “My Function:”, string
        return “yes”
(press return twice here to get the prompt back)
>>> print myFunction(“here I am!”)
My Function: here I am!
yes
>>> print myFunction
<function myFunction at 0x8176084>
```

Okay, so basically the Python interactive interpreter is your friend. It is great for quickly trying things out and getting used to Python's syntax. (Hmmmm, sound familiar? Kinda like a dot prompt? Or the Visual FoxPro Command Window?) And you get additional benefits from using the interactive interpreter with IDLE: namely syntax coloring and function argument ToolTip help. If you noticed, you learned several things in the above session: comments can appear on a line of their own or inline, a line can be continued with the backslash character, normal Python scripts can be imported into other Python scripts as modules, the basic print output functionality is adequate for quick testing, and functions are first class objects and thus can be passed to other functions. If you didn't get any errors importing the odbc module, you are ready to continue with this tutorial.

My goal is to introduce you to the simplicity and elegance of Python, and I'll do this by showing you how you can access and manipulate database tables from Python code. I'll walk you through the design of some test tables from VFP, accessing and manipulating the tables from VFP with the FoxPro ODBC driver, and then accessing and manipulating the tables from Python using the same ODBC driver.

First of all, make sure you have the Visual FoxPro ODBC driver installed (open the ODBC Administrator, go to the drivers tab, and locate the driver. Note the exact name of the driver, which you'll need when you enter the connect string later. My version is 6.01.8629.01, and the name is “Microsoft FoxPro VFP Driver (*.dbf)”. If yours is named differently, you'll have to substitute your name in my examples.

Time to create the table. Fire up Visual FoxPro, and follow along in the command window:

```
mkdir c:\temp\pytest
cd c:\temp\pytest
create table menu (iid i, citem c(32), ;
  nprice n(8,2))
insert into menu (iid, citem, nprice) values ;
  (1, “Pancakes”, 4.50)
insert into menu (iid, citem, nprice) values ;
  (2, “Coffee Cake”, 2.89)
insert into menu (iid, citem, nprice) values ;
  (3, “Tofu Scramble”, 5.00)
use
* connect using ODBC:



cString = [driver=] ;
  + [{Microsoft FoxPro VFP Driver (*.dbf)};] ;
  + [SourceType=DBF;SourceDB=c:\temp\pytest\]

iHandle = “c:/temp/pytest/"

* iHandle should contain a postive value - if it
* returns a '-1' value, it means the command
* failed
=SQLEXEC(m.iHandle, “select * from menu”)
BROWSE
```

If you see your three records in a sql cursor called “sqlresult”, then all is as it should be. Time to try the same thing from Python.

Before going on, I should back away at this point and explain that Python consists of a core that is extensible by importing external modules. These modules may have been distributed with Python, or they may have come from one of hundreds of developers worldwide, or they may have been written by you. The point I'm making here is that, unlike VFP, Python does not load up a lot of baggage that you may or may not ever use, but instead leaves it up to you to import the modules that you require. Indeed, you get to choose the modules that work best for you, instead of relying on one vendor to make the one correct choice for you, or refusing to provide functionality that you need.

Accessing databases requires the importing of an appropriate driver for the database in question, and to make it a less low-level of an experience I've borrowed a recipe from The Python Cookbook called 'LazyDB', written by John B. Dell'Aquila, and wrapped it with 2 layers of my own modules, one that defines a dbAccess baseclass, and one example subclass that can be used to access VFP tables. Other subclasses could be made to access other backend databases, such as MySQL or Microsoft SQL Server.

Anyway, there are 3 files in the download section that you need to save for the tutorial to work. They are: lazydb.py, dbaccess.py, and dbaccess_vfp.py. Save them to c:\temp\pytestcode\. dbaccess_vfp.py contains a subclass of dbAccess, which is defined in dbaccess.py. dbaccess.py instantiates the Connection class as defined in lazydb.py.

Take a look at these three files in a text editor (hint: open IDLE, then File/Open, and view the files that way, as you get the benefits of syntax coloring). Note that every script except for lazydb has an import statement. dbaccess_vfp imports the odbc driver, because that is how to connect to VFP tables. For my projects, I actually use a dbaccess_mysql script that, instead of importing odbc, imports MySqlDb which connects directly to MySQL without the need for ODBC. Both the MySQL and the VFP scripts access the common dbaccess and lazydb modules.

Anyway, take a look at the code in these three files – as you are new to Python, you won't completely understand it, but you should notice that the code is readable and the class definitions are simple. Notice at the bottom of dbaccess_vfp.py, there is a block of test code. This is, in effect, a way of including unit testing with your modules. The code will only execute when run as script (by typing 'python dbaccess_vfp.py' at the DOS command line), but will not execute when imported from another file. It is very similar to the convention good VFP developers have adopted of putting unit test code in their program classes that will execute when you run the code as a program, but not when you instantiate a class from that program file. The difference is that in VFP, you would put such test code at the top, but in Python it must go at the bottom. Just for kicks, why don't you run dbaccess_vfp.py and see if everything is working correctly. Open up a DOS command line (cmd.exe) and do the following:

```
c:
cd c:\temp\pytestcode
python dbaccess_vfp.py
```

If you saw the test output along with the 3 menu items, everything is working just fine. If you got a TraceBack message (Python's default error handler), you probably need to edit dbaccess_vfp.py to change the connect string to conform to your odbc setup. It is hardcoded in there – not normally a great practice but it does just fine for this example. Change it appropriately to match your setup. If you got a Windows message that the python executable could not be located, then you need to manually add the Python 2.2 program directory to your system path. Alternatively, if the .py extension got registered correctly, you can probably get away with simply typing:

```
dbaccess_vfp.py
```

Okay, so the script runs, proving that you've gotten data out of the VFP table. It is more fun to interact with tables, so it is time to start IDLE again and follow along in the Python shell. I'll be importing the dbaccess_vfp.py module, which is still sitting in c:\temp\pycode. Because it isn't in Python's search path for modules, I'll need to add the path manually. While there are other ways to achieve the same end result, this one introduces you to a basic Python type: the List.

Python is particularly well-suited for dealing with data because of its List, Tuple, and Dictionary data types. These are all sequences, and sequences can contain other sequences, resulting in a powerful matrix of possibilities. A list is a mutable sequence of items, and is specified by using square brackets ([]). Contrast this to a tuple, which is an immutable sequence of items. A tuple is designated by normal parens -

(). Lists are a great way to hold 'records', whereby each 'record' is another list of 'fields'. Anyway, the pathing issue:

```
>>> import sys  # basic system functions
>>> print sys.path
(I omitted the results, but do this yourself,
 and you'll see a list of strings)
>>> # append our path:
>>> sys.path.append("c:\\temp\\pycode")
```

Now that you have the path set up (did you notice that the list is actually an object with an append() method?) you can follow along with the following interactive session to read from and manipulate the test data:

```
>>> # import into the global namespace:
>>> from dbaccess_vfp import *
>>> # get a recordset:
>>> menu = dbVFP().dbRecordSet("select * "
        "from menu")
>>> # show all the records (note the clean
>>> # iteration style):
>>> for item in menu:
        print item.citem, item.nprice
(two CR's to exit the iteration block)
```

You should see one record per line. Ok, time to have some fun by updating and inserting records:

```
>>> # this time get a persistent object reference
>>> # to dbVFP:
>>> vfp = dbVFP()
>>> vfp.dbCommand("update menu set citem = "
    "'Pancakes and Spam' where iid=1")
>>> # this time define a function we can call
>>> # over and over again:
>>> def showAll():
        menu = dbVFP().dbRecordSet("select * from "
           "menu order by iid")
        for record in menu:
        print record.iid, record.citem, \
                record.nprice

>>> # now when we change something, we can just
>>> # call showAll():
>>> vfp.dbCommand("insert into menu "
        "(iid, citem, nprice) values "
        "(4, 'Rice Cakes and Water', 14.50)")
>>> showAll()
```

Play around interactively for a while, and start experimenting with writing your own custom scripts. You may not be all that excited yet, because you've noticed that so far its been all text output, no familiar graphical user interface. This was to keep the tutorial simple, to introduce you Python by showing you one way to access FoxPro tables. Rest assured, there are many ways to create professional GUI applications with Python. Just so you don't feel totally left out in the cold, here is a little teaser to show you how easy it can be to add a graphical interface to your application (enter this code in the Python Shell interactively, or if you are feeling more sure of yourself, make it into a script and then execute it):

```
import sys, Tkinter
Tkinter.Label(text="Enough spam for ya?").pack()
Tkinter.Button(text="Bye", command=sys.exit).pack()
Tkinter.mainloop()
```

Python Resources

Python is supported by a community of enthusiastic volunteers, in academia and the IT industry, from around the world. A great place to start is at the official Python website, at http://www.python.org. There are plenty of tutorials and lots of documentation there to keep you busy learning this exciting language. There is also a mailing list/UseNet group (python-list@python.org / comp.lang.python) that ranges from newbie to ultra-technical and that I've found quite fascinating. There are a number of books available, of

which I can highly recommend Python in a Nutshell, by Alex Martelli, O'Reilly 2003 as a great desktop reference and Python Cookbook, edited by Alex Martelli and David Asher, O'Reilly 2002, which contains recipes (how-to's) from the Python community.

Python and Visual FoxPro can work well together in situations where you have a desktop application and a web application accessing the same data. The desktop application can be written in Visual FoxPro while the web application can be done in Python. It can be difficult to deploy a VFP app to the web because of the load VFP tends to put on the server – unless you have a dedicated server for your VFP web app, you'll probably be better served with a Python web app on an Apache web server. Or, if you find that you need to deploy your desktop application to platforms other than the latest version of Windows, you can consider rewriting it in Python, coupled with a graphical toolkit such as Qt or wxPython. When it comes to development tools, there are many alternatives out there, and in my observations Python is worth a look.

Paul McNett is an independent software consultant in Hollister, California. While designing desktop and web applications using Visual FoxPro is still his primary focus, he is also using developer tools from the open source community, including Python, to deploy to the OSX, Linux, and Windows platforms. He can be reached by email at p@ulmcnett.com or by phone at 831.636.9900.