

And now for something completely different...

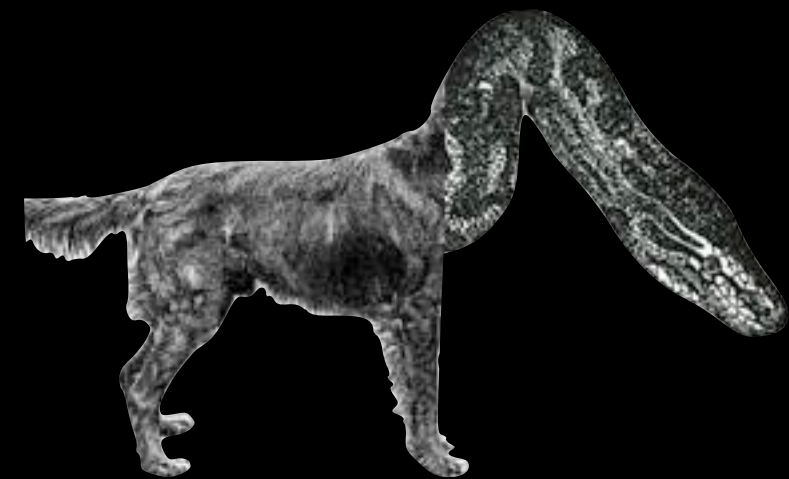
Kevin A. Mitchell
kevin@dashingfalcon.com
[@kamitchell](#)



Hi, I'm Kevin Mitchell, and I'm a Mac software developer. I've been programming for the Mac since oh, since it came out. Cocoa is the best application framework I've ever worked with and I've also been a big fan of the Python programming language for quite some time now.

PyObjC

- Objective-C and Cocoa Programming
- and Python
- Open-source project with long history, as far back as 1995.



When you bring those two together, you have PyObjC, an Apple-supported bridge from Python to the frameworks supported by Cocoa and Objective-C.


Wherefore PyObjC?

- Language evolution away from C-type languages
- Pointers are “too much rope”
- Memory management is cumbersome



Why PyObjc? C-type languages have a lot of features that are deeply rooted in their history. Pointers, for instance, are way too much rope. Mess up a pointer and you're asking for a crash. Memory management is done manually, and can be different for different libraries or frameworks. If you make a mistake there, the app crashes or leaks, or it has security problems. Objective-C 2.0 fixes the memory allocation issue with garbage collection, but it still has pointers. And on top of it all, header files require repetition.

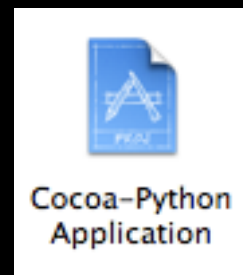
Wherefore PyObjC?

- Language evolution away from C-type languages
- Pointers are “too much rope” 
- Memory management is cumbersome
- Objective-C 2.0 has garbage collection but still has pointers
- Header files violate DRY

Objective-C 2.0 fixes the memory allocation issue with garbage collection, but it still has pointers. And on top of it all, header files require repetition. I've been wanting to do more of my coding in languages that don't expose pointers, and so I've been exploring PyObjC.

Use PyObjC to make...

- Applications (Xcode template)
- Plugins
- Foundation command-line tools



You can use PyObjC to build not only applications, but also plugins and command-line tools based on Foundation.

PyObjC Frameworks

- AddressBook
- AppleScriptKit
- Automator
- CalendarStore
- CFNetwork
- Cocoa
- Collaboration
- CoreData
- CoreText
- DictionaryServices
- FSEvents
- InputMethodKit
- InstallerPlugins
- InstantMessage
- InterfaceBuilderKit
- LatentSemanticMapping
- LaunchServices
- Message
- PreferencePanels
- PubSub
- QTKit
- Quartz
- ScreenSaver
- ScriptingBridge
- SearchKit
- SyncServices
- SystemConfiguration
- WebKit
- XgridFoundation

A vast collection of OS X Frameworks are available: Cocoa of course, and CoreData, LaunchServices, Quartz, SystemConfiguration, WebKit...

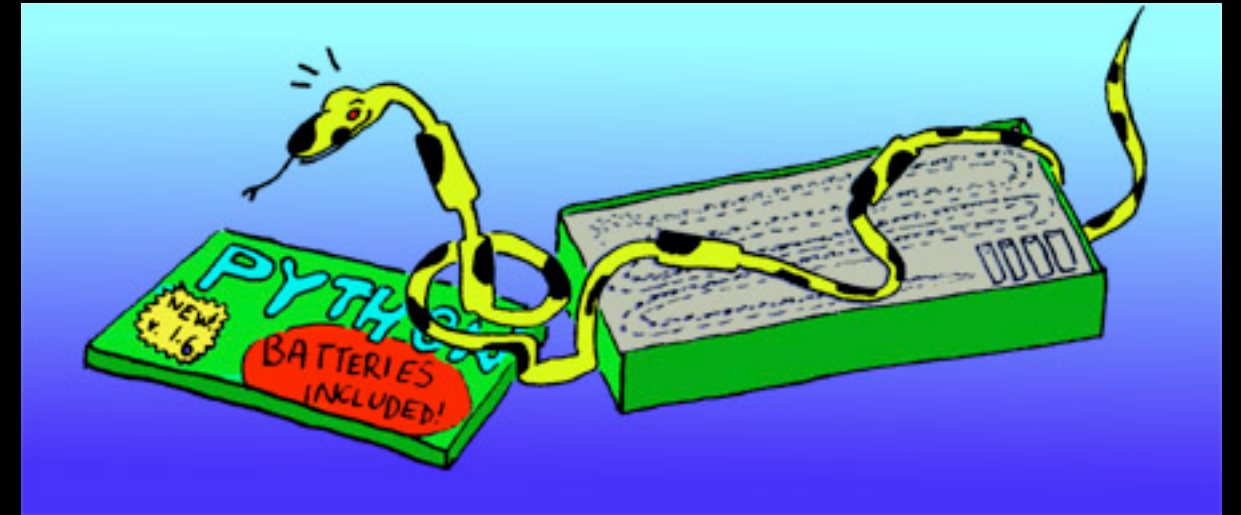
Cocoa Bindings

- Key-Value Coding
- Key-Value Observing
- Automatic KVO notifications

PyObjC has real support for the most modern features such as Cocoa Bindings. Coding and observing, and it'll automatically make KVO notifications if you change an attribute on an NSObject-based class.

Python advantages

- “Batteries included”
- Useful third-party modules, like feedparser and simplejson



Frank Stajano/Wikimedia Commons

When using PyObjC, you also get the advantage of the rich set of code available for Python. There are many built-in modules for everything under the sun, from file parsing to internet protocols. To add to your rapid development toolkit, there are many useful third party modules, like feedparser and simplejson

Script and extend apps

- Appscript: Python scripting is easier than AppleScript
- Application plugins that don't require developer tools

Take advantage of Appscript, and easily control other applications with Apple Events. Since Python is interpreted, write application plugins that don't require developer tools. Users can modify these plugins in the field.

Slow? No.

- Heavy lifting still done in native code.
- Future: What is the speed of an Unladen Swallow?
(European: 11 m/s or about 24 mph, actually)



photo by Jurvetson (flickr)

Python may seem a poor choice for an application language. Could it be too slow? No. Most of the heavy lifting is done by native code in the frameworks. Google's Unladen Swallow is a project to use LLVM to add a JIT and native code to Python.

Optimize the Python way

1. Write code in Python
2. Find the slow parts. Measure!
3. Rewrite those in ~~C or C++~~ Objective-C
4. Easy loading of native bundles (`objc.loadBundle()`)

And besides, the Python philosophy is to write code in Python, find the slow parts by measuring, and then recode them in C. If you rewrite the slow parts in Objective-C, then PyObjC makes it easy to load your native code bundles back into Python.

PyObjC and (Snow) Leopard

- Xcode 3.0+ support
 - Project templates and autocompletion
 - Alas, no full-screen debugging
- Interface Builder
 - Read and create class files

There's pretty good support for PyObjC in Leopard. There's project, editing and autocomplete support in Xcode, but alas, no debugging for Python code. Interface Builder has good support; it can create and read class files.

Debugging

- NSLog
- import logging

So what can you do for debugging? Well, there's always the NSLog function, or you can Python's standard logging module. For well-understood and smaller programs, these are probably good enough.

Debugging

- NSLog
- import logging
- pdb on the command line
- Eclipse and PyDev, with a few tricks (CFProcessPath)

If you're interested in stack traces, stepping, and local variables, you can start up your app with the Python Debugger, pdb. If you like a source view during debugging, you can use Eclipse and PyDev. You just need to tell Foundation the location of the “executable” so it can find the pieces of the bundle.

RandomAppPy

```
from Foundation import *  
from AppKit import *  
import objc  
import random
```

What does a PyObjC application look like? This is the RandomApp sample from Hillegass' book redone in Python. It imports Foundation and AppKit. The objc module provides some helpers for the bridging, and for random numbers, this version uses Python's random module

RandomAppPy

```
from Foundation import *
from AppKit import *
import objc
import random

class Foo (NSObject):
    textField = objc.IBOutlet()

    def awakeFromNib(self):
        now = NSDate.date()
        self.textField.setObjectValue_(now)

# continued...
```

Cocoa class declarations are done the Python way, just inherit NSObject. textField is declared as an outlet, and that part of the code was generated by Interface Builder. The next bit of code sets up the textField when the nib is loaded.

RandomAppPy

```
# ...class Foo continued
@objc.IBAction
def generate_(self, sender):
    generated = random.randint(1,100)
    NSLog("generated = %d", generated)

    # Ask the text field to change what it is displaying
    self.textField.setIntValue_(generated)

@objc.IBAction
def seed_(self, sender):
    # Seed the random number generator with the time
    random.seed()
    self.textField.setStringValue_("Generator seeded")
```

The Python decorator defines these two functions as actions; that part was generated with Interface Builder too. You can see the PyObjC rule for naming methods, just change the colons to underscores and glom together to form the method name.

Interactive AppKit

```
$ ipython
Python 2.5.1 (r251:54863, Feb  6 2009, 19:02:12)
Type "copyright", "credits" or "license" for more information.

IPython 0.9.1 -- An enhanced Interactive Python.
?          -> Introduction and overview of IPython's features.
%quickref  -> Quick reference.
help       -> Python's own help system.
object?    -> Details about 'object'. ?object also works, ?? prints more.

In [1]: from AppKit import *

In [2]: w = NSWorkspace.sh TAB
NSWorkspace.sharedWorkspace  NSWorkspace.shouldUnmount_

In [2]: w = NSWorkspace.sharedWorkspace()

In [3]: w.absolutePathForAppBundleWithIdentifier_(u"com.apple.iTunes")
Out[3]: u'/Applications/iTunes.app'
```

ipython is a superb interactive shell that naturally works with PyObjC. Tab-autocompletion helps with the identifier names and saves typing while trying out a method on NSWorkspace. Try out framework calls without compile cycles or a debugger!

PyObjC in the wild

- Checkout, a point-of-sale app: <http://checkoutapp.com>
- Google Quick Search Box takes plugins written in Python: <http://code.google.com/p/qsb-mac>

You can find PyObjC in the Checkout point-of-sale application. Google's Quick Search Box, which is the descendent of Quicksilver, accepts plugins written in Python.

More

- pyobjc.sourceforge.net
- Discussed on pythonmac-sig: <http://bit.ly/pythonmac-sig>
- <http://developer.apple.com/cocoa/pyobjc.html>

You can find more information about PyObjC at pyobjc.sourceforge.net, where there are some mailing lists, and it's also discussed on pythonmac-sig. I hope you will find PyObjC interesting, and thank you for your attention.