Python for iPhone & iPad

Research Article

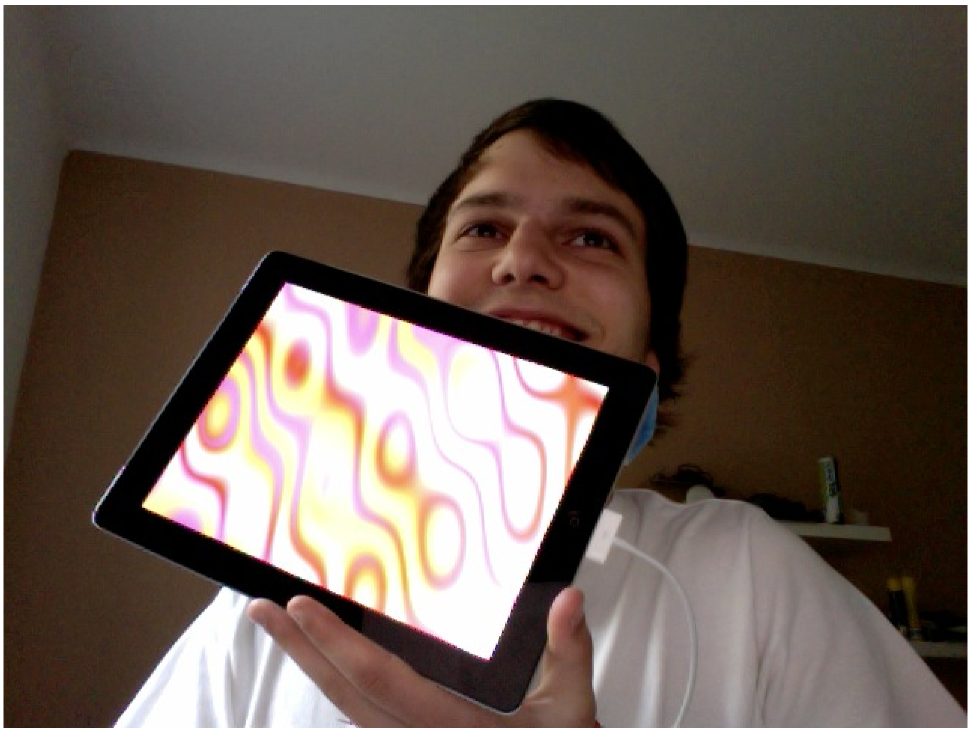
Christopher Denter

cdenter@uni-koblenz.de

2011

This document is a preview/draft for the online Python and iOS communities.

These images show the results of my research. The first depicts a simple interactive application where users can draw lines with their fingers, the second shows a simple OpenGL fragment shader effect, controlled via Python.



CHAPTER

ONE

PYTHON FOR IOS

Applications for Apple's iOS devices are primarily developed using the languages Objective-C(++), C or C++. The apparent lack of support for other programming languages can easily be explained by the fact that Apple once banned most higher-level languages. As a matter of fact, this restriction has since been lifted and it is now allowed to execute most programs as long as no additional code is downloaded at runtime (this still bans most Flash programs as encountered in web browsers).

This relaxation poses an interesting research opportunity to explore how well a high-level language such as Python can be tuned to nicely fit into the restricted ecosystem that is provided by a device such as an Apple iPad [1].

The goal of this research is to determine the feasability of using Python for the major part of an application's codebase, taking into account the special requirements imposed by iOS devices. It will be proven that iOS applications can be built with Python by providing a set of interface wrappers around the necessary system APIs as a proof of concept. The application would then embed the Python interpreter and execute Python code at the core that uses the interface wrappers to communicate with the system.

1.1 Compiling Python

Understandably, there is no Python distribution for iOS devices, so one needs to compile the interpreter oneself. Given the numerous differences in the architectures of the iOS systems, this by itself is a non-trivial task and requires the interpreter to be patched. Having applied these patches, the interpreter can be cross-compiled for the ARM architecture (as used in iOS devices) and deployed to an actual device [2]. One can then proceed with embedding the interpreter.

1.2 The Framework

The ultimate goal of any Python developer in this context would be to be able to simply write a piece of Python code (and nothing else) and run that on an iOS device.

While this is definitely achievable, additional work needs to be done behind the scenes. In particular, any application that wants to run on an iOS device will need to have a C/ObjC framework which is used by the system to start the application. What we're looking at, thus, is an Objective-C application with an embedded Python interpreter. The ObjC part will be the interface between the Python code and the system, receiving and passing on such events as touches or memory warnings.

[1] Developments of this kind were previously only possible by 'jailbreaking' the device. To the best of my knowledge, no proper solution for application development using Python on iOS exists at the time of this writing.

[2] Going into the details of what needed to be changed in the Python source code in order to be able to run Python on iOS devices is well beyond the scope of this article.

1.3 Embedding Python

Python has long since supported being embedded it into other applications. In this context, we're especially looking at two approaches to using Python as part of another program:

- Embed Python as usual, handing off control to the interpreter in some places in order to have it perform operations that are limited in time.
- Embed Python but hand off control over the entire application's life cycle to Python exclusively. In particular, this means keeping the main run loop in Python code.

Both of these approaches have advantages and disadvantages. The first approach is pretty easy to realize once a working Python environment has been set up. It is possible to use Python in pretty much arbitrary places in an existing Objective-C program. It does not, however, easily allow developers to write entire applications to be executed on iOS devices that are solely written (at least from the developer's perspective) in Python.

1.4 Utilizing an Existing Cross-Platform Toolkit

The second approach makes this easier: We can take an existing cross-platform Python GUI toolkit [3] that already in its design incorporates the ability to run on different platforms via abstraction. We then provide an implementation of these abstract pillars (e.g. opening a window, rendering text, images, videos, etc.) that allow the toolkit to run on a new platform such as iOS. One can then take any well-written application based on that toolkit (and not using unsupported third-party dependencies) and run it using the prepared environment for iOS.

What one has to do is rather simple: The iOS environment contains a folder named 'YourApp/' in which one simply places the files and folders of the application. In that folder, there needs to be a 'main.py' file that is used as an entry point to the application. It is this file that will be executed by our iOS framework. The application then simply uses the feature set of the GUI toolkit which, internally, use the specific iOS implementations. This makes the application portable to different platforms as well.

1.5 Results & Conclusion

Suitable applications written using the GUI toolkit [3] run well on an iPad 2. Given that the toolkit exclusively uses OpenGL ES to render graphics and provides its own set of widgets, no attempt was made to use native iOS UI widgets. This does make the applications look alien to iOS users, but allows for a consistent look of the application across platforms.

As this is only a proof of concept, there is still a lot of work to do before one can hope to deploy an application built this way using facilities such as the Apple App Store. The most important points would be to radically minimize the size of the application, support orientation changes, respond to low-memory warnings, support resuming, and many more. Even if the aforementioned points are addressed, it remains to be seen whether Apple would allow such an application in practice.

1.6 Acknowledgements

This research has been carried out at the University of Koblenz-Landau under the supervision of Prof. Dr. Stefan Müller. Thanks go to the Kivy team and Mathieu Virbel in particular who helped with the port of the toolkit.

[3] For the purpose of this paper, the kivy framework has been used and extended. It already supports Windows, Linux, Mac OS X and Android.