

Johannes Woolard
Oxford University
February 28, 2008

CRUNCHY: CRUNCHING ON PYTHON DOCUMENTATION

Abstract

I describe Crunchy, a Python program that André Roberge and I have created. Crunchy acts as a proxy while browsing HTML-pages over HTTP and embeds “widgets” in the HTML code forwarded to the browser. These widgets allow interactive execution of Python code.

I will discuss the general system, its capabilities and its security implications. In addition, I will give a summary of our plugin architecture.

This short article accompanies a talk I will be giving at Pycon 2008 in Chicago.

Introduction

Crunchy is a system for adding interactivity to Python documentation. This documentation can be in the form of HTML or (in the newest release) ReST. ReST is handled by converting it to HTML. For the remainder of this article I will discuss how Crunchy interfaces with HTML. Crunchy can work in one of two modes: it can automatically recognise code examples and make them interactive (i.e. allow them to be edited and executed) or it can follow directions placed in the HTML code using a markup language we call VLAM.

In 1925, C. J. Jung (the psychologist) said:

“Even the best methods of conscious education can sometimes be completely nullified by bad examples.”

Crunchy’s aim is to make documentation clearer and tutorials more engaging through the use of good examples. In Crunchy, Python examples can be tried directly in the documentation, they can be modified and executed. We believe that this creates a better understanding of the material than traditional examples that need to be copied over into an external program to be tried out.

History and Capabilities of Crunchy

The project, which eventually became Crunchy, was started in the spring of 2006: initially André Roberge designed a demonstration program that loaded HTML files from the local file system and embedded Python interpreters and editors. This initial system was based on CherryPy.

I joined the project through the Google Summer of Code¹ 2006. By the end of summer 2006, André and I had brought Crunchy to a point where it was generating a lot of interest from the community. Since then, development has proceeded gradually. Crunchy was presented at Pycon 2007 in Dallas, Texas where it attracted a reasonable amount of attention. As a result of some comments made to André at Pycon 2007, we designed and implemented a simple plugin mechanism (more on this later).

¹<http://code.google.com/soc/>

Over time, Crunchy has gone through a few names; initially it was known as IT (for Interactive Tutorials - as suggested by André's daughter, Evelyne), then it was Crunchy Frog (after the Monty Python Sketch) and finally it became Crunchy (when we discovered that Crunchy Frog was already taken).

As mentioned above, Crunchy is effectively a proxy for HTML pages served over HTTP. Crunchy is implemented as an HTTP server (using the Python standard libraries - there are plans to move over to twisted at some point in the future). Crunchy can take an HTML page and embeds widget. These widgets can communicate with the server using COMET. The following widgets are currently supported by Crunchy:

- Interpreter: A Python interpreter very similar to the command line version or IDLE.
- Editor: An area to edit Python code and facilities for executing it, with stdin and stdout redirected to the webpage.
- Doctest: Displays some doctests² and invites the user to write Python code to satisfy those tests.

In addition, a graphics canvas is supplied and available from both the interpreter and the editor. Simple graphics can be created and viewed directly in the web page.

Interactive Widgets

In order to execute arbitrary Python code from a web page, the code has to be fed to a Python interpreter. There are a number of ways of doing this: one would be to write a fully featured Python implementation in Javascript, another would be to embed a Python interpreter in the browser³ and a third is to send the code to a server and execute it outside of the browser. We use this last approach. The same HTTP server that is used as a proxy for HTML pages also accepts specially formed HTTP requests: When the user wants some code executed, it is sent to the server (using a specially formed POST request) and executed in the server (using Python's exec statement).

Obviously there is a need to send information from the server to the webpage, but we are limited by HTTP being a strictly client-server protocol (only the client can initiate communication). In older versions of Crunchy, the webpage would poll the server at regular intervals to see if there was any new information, but this was extremely inefficient and led to high processor usage, even on modern machines. Our solution to this is to use "long polling", a variation of COMET: The client initiates a connection, and the server doesn't send anything down it until the client is ready.

Security Concerns

Clearly, because we run arbitrary Python code sent to an HTTP server, there are security issues. We need to be able to guarantee that only code explicitly cleared by the user can be run. Firstly, we get around this by binding the server to a loopback adapter on the computer (i.e. 127.0.0.1). Now we only have to worry about attacks running on the local host. We get around this by generating a unique session key that is only given out to pages served from the server that are allowed access to it.

We must also consider the possibility that malicious Javascript in the page could send code to the server: we get around this by stripping all pre-existing javascript out of the HTML files that we

² <http://en.wikipedia.org/wiki/Doctest>

³ This has been done using Microsoft Silverlight and IronPython

process. This of course means that some webpages not function correctly, but we consider this a reasonable sacrifice.

It is of course still possible that someone could brute force the session key, or that an attacker could pose as a legitimate user and download a page containing the session key. These attacks could then be used for privilege escalation.

Plugins

In order to make it as easy as possible to extend Crunchy's featureset in the future, we have implemented a plugin architecture. Crunchy is essentially divided up into two sections: The first being the core, which provides the server and a few other basic services, and the second being a set of plugins. These plugins implement each of the individual widgets.

Each plugin is encapsulated as a `.py` file (i.e. a Python module) and placed in a special folder. At startup this folder is scanned and all the plugins found are imported and initialised. Plugins may depend on other plugins, and they are automatically sorted into the correct order⁴. All plugins must provide a `register()` function so that they can be initialised.

Our plugin system has been designed to be extremely simple - new features often only need a few lines of code, and even the extremely complex functionality of interpreters is implemented in a few tens of lines.

Conclusion

Crunchy is an innovative way of viewing programming language documentation. In many ways it is similar to literate and elucidative programming - but it is aimed specifically at documentation. Over the next few months we expect to release version 1.0 - and we would be grateful for any help the community can give us.

To download Crunchy, please visit <http://code.google.com/p/crunchy>, and feel free to join the crunchy-discuss mailing list advertised on that site.



This document is licensed under a creative commons license.
See <http://creativecommons.org/licenses/by-sa/2.0/uk/>

⁴This is a form of Topological Sorting, and as such, fails if there are any dependency cycles. A suitable error message is displayed.