The Python Papers

Volume 2, Issue 2
pythonpapers.org

Journal Information

The Python Papers

ISSN: 1834-3147

Editors

Referencing Information

Articles from this edition of this journal may be referenced as follows:

Author, “Title” (2007) *The Python Papers*, Volume N, Issue M, pp. m:n

e.g. Maurice Ling, “Firebird Database Backup by Serialized Database Table Dump” (2007) *The Python Papers*, Volume 2, Issue 1, pp. 7:15.

Copyright Statement

Referees

None for this issue.

The Python Papers

Volume Two, Issue Two : May 2007

In this Edition:

includes it's own CherryPy web server and provides a single page application with the features and interface one would expect from a standard desktop application. Although it is not unique in it's use of AJAX or it's use of a python based web server, it is unique in that it is a completely asynchronous interface which looks and behaves like an application rather than a web interface.

EuroPython 2007 Pages 33-35

Paul Boddie, David Boddie and other EuroPython2007 Wiki Contributors

Now in its sixth year, EuroPython - known more formally as the European Python and Zope conference - moves on once again. Upholding the EuroPython tradition of helping European Python users and others to learn more about Python and its uses, the conference is also a great opportunity to become acquainted with different parts of the European continent. This year, EuroPython descends upon Vilnius, the capital city of Lithuania, calling it home from Monday 9th July until Wednesday 11th July 2007. Although the programme for this year's conference has yet to be finalised, reports and materials from last year's highly successful event may be persuasive for those considering making EuroPython 2007 part of their schedule.

Python Events Page 52

A list of upcoming Python events.

Peer Reviewed Submissions:

ctypes. ctypes run! Pages 36-41

Alex Holkner

One of the new features of Python 2.5 is the introduction of ctypes as a standard library module. At the simplest level, ctypes adds the standard C types to Python: signed and unsigned bytes, shorts, ints and longs; as well as structs, unions, pointers and functions. At run-time it can load a shared library (DLL) and import its symbols, allowing a Python application to make function calls into the library without any special preparation. ctypes can be used to wrap native libraries in place of interface generators such as SWIG, to manipulate memory and Python objects at the lowest level, and to prototype application development in other languages.

A Rails / Django Comparison Pages 42-51

Ben Askins and Alan Green

Ruby on Rails is the dominant web programming framework for Ruby, and, even outside the Ruby community, is considered the epitome of the latest generation of high-productivity, open source web development tools. Django is one of many competing web development frameworks for Python.

In this paper, Askins and Green compare the two frameworks from the point of view of a developer attempting to choose one of the two frameworks for a new project.

The Python Papers' Review Policy Pages 53-54

The Python Papers review policy covers both industry and peer-reviewed articles. This is included for academic purposes.

Letter from the Editor

Tennessee Leeuwenburg

Hello to the readers of *The Python Papers*! This issue marks a major landmark in our publication, with the largest number of articles ever, the greatest number of contributors ever, the results of our first competition ever and the most downloads ever! May this trend continue!

In this issue, we present a number of industry articles. These include *Python in Education* and *MPD* WebAMP, as well a great insight into Python in Germany, a wrap-up of PyCon 2007, a preview of EuroPython 2007 and a look at some great videos prepared by primary school students. Our peer-reviewed section reproduces two selected papers which were originally presented at the Open Source Developer's Conference 2006[1] (Melbourne, Australia).

The past few months have been a great time for *The Python Papers*. Since the last issue, we have had over 3, 000 visitors to the site. One of the goals of the journal has been to appeal to a global audience. As can be seen in Illustration 1, the readership is based very widely indeed. The largest visitor group by continent is Europe, with 34% of site visits. North America is second with 28% and Asia is third with 16%. Oceania / Australasia accounts for 13%, South America 4% and Africa 2%. 3% were not accounted for. These statistics were taken on May 20[2].

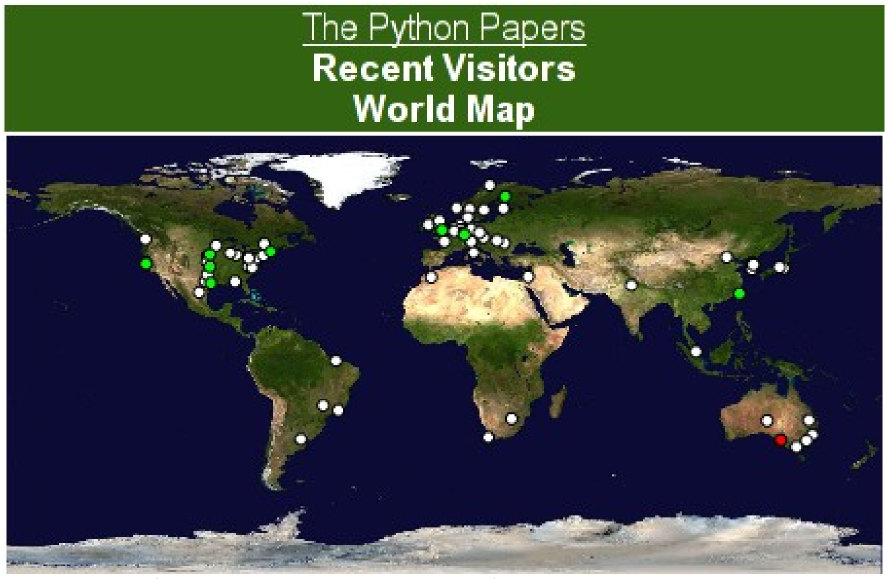


Illustration 1: World map showing the last 100 visitors to pythonpapers.org on 20 May 2007

I'd like to express how pleased I am that we have been able to engage so many distinct groups. I am not aware of any detailed investigation of the use of Python internationally, so this may represent unique information on the spread of Python around the planet. If anyone from Japan, Singapore, Taiwan or Israel would like to contact *The Python Papers*, we would love to hear from you as we do not currently have any contacts from these countries. Indeed, we would love to hear from anyone who would like to be involved with this journal.

Another exciting discovery has been the audio transcript of *The Python Papers* which is being distributed by scribd. The previous edition is currently available from http://www.scribd.com/doc/6424/The-Python-Papers. From here, the journal may be viewed as a PDF online, as MS Word, plain text or a text-to-speech rendering may be downloaded as an MP3 file. Having listened to this audio, I must congratulate that site on its great quality. The only real drawback is that the opening pages of the magazine, the table of contents and the code snippets can seem repetitive. The article text, however, comes across wonderfully.

There have been significant growing pains involved in the development of *The Python Papers*. The editorial board, especially yours truly, has been struggling to achieve all the goals that would ideally be met. We would love to hear from anyone who would be happy to give us a hand,

1 http://osdc.com.au/
2 http://www.sitemeter.com/?a=stats&s=s28pythonpapers&r=81

whatever your area of expertise or interest may be. In particular, we would love to hear from one or more people who are happy to take on the management of our website. This would include not just the technical aspects of creating a blog (working with the other editors to agree on a front page design and other "getting started" issues) but also ongoing maintenance of the website, including helping to create new documents as needed, respond to web queries etc.

There are a number of other roles which a committed individual could adopt and thereby provide enormous assistance. A representative for each country would be an enormous help, especially in collating contact and web presence information on local Python User Groups and events. It is difficult to keep abreast of the variety of activity around the world. Local assistance here would be a huge help.

Contribute and win – prizewinner announced!

Congratulations to Chris Seickel! He is the lucky winner of the "contribute and win" competition. He will receive a 3-OS license for Wing IDE, a fantastic development environment created by Wingware[3].

Introducing The Team

Tennessee Leeuwenburg – Editor-in-Chief

Tennessee Leeuwenburg is a software developer working at the Australian Bureau of Meteorology on automatic text generation of weather forecasts. Prior to this he spent time working on meteorological data transfer standards in the form of the OpenDAP database system.

Maurice Ling – Associate Editor

Maurice Ling is a PhD candidate in the department of Zoology of The University of Melbourne working on text analysis of biological literature for the purpose of understanding hormone interactions in the mouse mammary cell.

Richard Jones – Associate Editor

Richard is Blue Box Device's lead OpenGL developer with over 10 years of broad experience working with multiple languages and tools in web-based management systems, data archive, meta-data systems, computer graphics, business systems, e-commerce and communications. He also runs the bi-annual PyWeek Game Programming Challenge and is involved with the organisation of the Open Source Developer's Conference in Melbourne, Australia.

Stephanie Chong – Associate Editor

Stephanie is currently studying Arts/Law at The University of Melbourne.

Contacting The Python Papers

The editors may be contacted via email at: editor@pythonpapers.org. We are always happy to receive feedback, suggestions for improvement and ideas for future articles and topics.

Contribute to The Python Papers

If you would like to contribute an opinion piece, an article, participate in an interview or submit a paper for review and publication, please don't hesitate to contact us at editor@pythonpapers.org.

3 http://wingide.com/

Python User Group Highlights: Introducing pyCologne

Christopher Arndt and Rex Turnbull

setup(name='pyCologne', description='Python User Group Köln')

What do you do, if you're new in town, a jobless software developer and a Python fan? Right, you set up a local Python User Group. That's at least how it worked out for me when, last fall, I suggested a meeting of Python enthusiasts at a local Linux fair and nine other like-minded people showed up on the spot. Thus the Python User Group Cologne, quickly afterwards dubbed *pyCologne*, was born and we've had regular meetings every month since then.

For those readers unfamiliar with Germany, the city of Cologne (the proper German name is "Köln"), is by number of inhabitants the fourth biggest city in Germany, with just over one million citizens. It has a strong industrial tradition, being to the south-west of the highly industrialized and populated area of the "Ruhrgebiet". Today it has a strong footing in the tertiary sector, with the media (tv) and design industry being major sources of employment. I'll give you some more information about Cologne and its sights later, let me tell you something more about our group first.

def pyCologne():

Given Cologne's geek-friendly environment and a large university, it is perhaps not surprising that our group's meetings were able to attract 15-18 people each time so far and that there are about 50 subscriptions to our mailing list[4]. We have no explicit membership but there is a "core" of about 8-10 people who show up at every meeting. Since the group is still rather young, it remains to be seen how things will develop in future.

As things stand now, **pyCologne** meets on the second Wednesday every month. Our meetings begin with a "business" part with one or two presentations or discussions and last about two hours as the content always leads to more discussions. After the official part, we usually head off to a pub for food, drinks, and friendly conversation.

PyCologne User Group Members

4 https://lists.uni-koeln.de/mailman/listinfo/python-users

x.get_interests() for x in pyCologne.items()

There are indeed plenty of interesting, Python-related topics to talk about, since the members of our group use Python for an astonishingly wide area of applications. Some are directly involved in developing software professionally that makes use of Python in one way or another, others just use it as a tool for internal purposes. I'll mention just a few fields where our members employ Python (I could easily list a dozen more):

- Grid-computing (for example in the aerospace industry)
- Game scripting
- Web development
- Data-mining
- Financial transaction systems in the banking business
- Zope and Plone
- Personal tools and system administration

There are also a few of us who have little or no programming experience and want to find out more. Though it is true that Python is still relatively little known in Germany, interest in it is developing fast. The past months have seen the founding of several other German PUGs besides pyCologne. Some well-established projects from the Python world were also founded by German developers, for example the **MoinMoin** wiki system[5] (which also drives the official Python wiki[6]) and **Stackless Python**[7], an implementation of the Python interpreter that doesn't use a global interpreter lock (GIL) and features high-performance, concurrent computing mechanisms.

Cologne.__doc__

Cologne is not far from the **Neanderthal**, near Düsseldorf, where the famous remains of the ***Homo neanderthalensis*** were found, the species of ***Homo*** which inhabited Europe prior to ***Homo sapiens***. People have been living in the area besides the river Rhein since then – a rather long time. Cologne itself got its name from a Roman settlement, ***Colonia Claudia Ara Agrippinensium***, that was founded around 40 AD – which makes it one of the oldest, continually inhabited cities in Germany.

Night-time view accross the Rhine towards the old town and the Dom. http://commons.wikimedia.org/wiki/Image:Koeln_Nachtaufnahme.jpg, GNU FDL

Though much of the city center of Cologne was destroyed in World War II – a fate that Cologne shares

5 http://moinmoin.wikiwikiweb.de/
6 http://wiki.python.org/moin
7 http://www.stackless.com/

with most German cities - a good part of the old town was preserved or rebuilt and is now a favourite tourist haunt. The inner town is dominated by Cologne's landmark -- the **Dom** (Cathedral) with its twin spires, work on which was begun in 1248 and finally finished in 1880. It is the tallest gothic church in the world and can be seen for miles. The city's citizens are rightly proud of this historic building.

Kölsch, the local beer variety, can be found in all restaurants and bars in and around Cologne. It is a clear, yellow-colored, top-fermenting beer and is served in small beaker-like glasses, where the beer stays cold and fresh. When going out for a drink in Cologne, it is easy to rub shoulders and have a chit-chat with the locals, who are said to have a friendly, out-going nature.

finally:

Meeting many other people who like Python is a fun and interesting experience and we are always interested in hearing from others sharing our interest in all things Pythonic. Whether you live in the region or you are just staying here on a business or holiday trip, why don't you check out our website on the German Python wiki[8] [6]_ and see when we have our next meeting? The website also has slides and examples from the presentations given at past meetings and some more photographs. If you have your own Python user group and you want to share experiences, we would love to hear from you as well, wherever you are!

And, by the way, in case you wondered, thanks to the many contacts made via our group, I am now working as a freelancing Python software developer...

To find more information, or to contact members of PyCologne, visit *http://wiki.python.de/pyCologne*

8 http://wiki.python.de/pyCologne

Python in Education

Michael Tobis

SALIENT QUOTATIONS

"*Should the computer program the kid, or should the kid program the computer?*"
- Seymour Papert

"*We want to explore the notion that virtually everybody can obtain some level of computer programming skills in school, just as they can learn how to read and write.*"
- Guido van Rossum

"*Python makes too much sense as a learning language to not ultimately have an effect in the education arena.*"
- Arthur Siegel

"*If schools won't accept classes that have Python in them, then it doesn't matter how much better Python is than BASIC, the classes cannot be taught.*"
- Dean Nevins

PYTHON IN EDUCATION

If learning is about information, then surely education will change somewhat under the influence of pervasive information technology.

This is an exciting time for those of us who believe that software development in general, and Python in particular, has a role in general education. Three of the four keynotes at PyCon '07 involved the idea of computing as a medium for education and/or communication. Adele Goldberg talked about the frustrations of getting carefully designed computational curriculum into the public school systems. Robert "rOml" Lefkowitz talked about code as a communication medium. Finally, on a more practical note, Ivan Krstic discussed the very exciting prospects of the One Laptop Per Child project, whose modest goal is to make computing available to all children everywhere.

These discussions gave a boost to the quiet world of educational Python. It seems like a good time to take stock of the entire educational programming landscape and Python's role in it.

COMPUTER LITERACY

In the early days of computing, it became apparent that computers would become ubiquitous and part of daily life. At that time, we had little idea what shape they would take in a generation.

I recall an episode of the futuristic TV news program "The Twenty-First Century", broadcast in 1967 hosted by the renowned Walter Cronkite. It discussed the implications of home computers. A professor's family had a massive mechanical teletype (a terminal that used a typewriter mechanism rather than a CRT or LCD display) in their home. The demonstration application was, as I recall, changing the proportions in a recipe so that it served seven rather than four.

Presumably, it was a great tangle of Lisp under the hood, and awkward for the user, but it appeared that if you persevered, a recasting of your recipe (with all the quantities multiplied by 7/4 and converted to standard cooking units) loudly clanked out onto the fanfold paper. At the time, the 21st century was promoted in the media to be an almost flawless utopia, but even so one had the sense that both Cronkite and the professor's wife were struggling to express any enthusiasm for the concept of home computing.

If you think about what "computer literacy" means today in high schools, trade schools and nontechnical programs in university, though, you'll see that it amounts to recipe scaling writ large. Mass market software presents complex and often counterintuitive user interfaces, and the skill of navigating through each of them is valued and promoted. The expression "computer literacy" is taken to mean a capacity to cope with mass market application software. (If this were actually useful in schools, perhaps it would work better if the adolescents taught the faculty.)

PROGRAMMING FOR CHILDREN

Of course, we do need to be able to cope with applications. Still, there have always been two visions of computing, one as a world in which there is a very sharp distinction between producers and consumers of software, and another where the continuum is blurred.

The view of computer literacy as a continuum which includes programming was formed by the early users of computers. In those days, there were no "applications". Being a computer user amounted to being a programmer. Many of us felt that the skills we learned by wrestling with the structure and logic of a program would be applicable to many aspects of life. We felt that the advantages we were getting could be democratized. In this view, "computer literacy" was about coding, not about being able to push a mouse around and click on things effectively.

This view is succinctly summarized by Georgia Tech Professor Mark Guzdial's phrase on a blog page: "computing FOR learning (as opposed to learning ABOUT computing"[9].

Seymour Papert (with Willy Feurzig) developed Logo[10], the original “learning language” in the late 1960's. The ideas of Logo were explicitly based on developmental psychology[11]. Papert not only claimed that remarkably young children (fourth graders, about 9 years old) could be exposed to the ideas of programming, he was successful in demonstrating this. Many subsequent teaching environments have been based on Logo, and especially on the observation that "turtle graphics" were a uniquely compelling development target for novices. (It's interesting to contemplate the fact that it was less expensive to construct a mechanical turtle than a graphic display in those days!)

Alan Kay's remarkably prescient 1971 "Dynabook" paper[12] is a remarkable exposition of the possibilities of computing from a time long before anyone actually had so much as a portable display. Kay and Papert are still with us as I write (although, unfortunately, Papert was seriously injured recently). Among their recent efforts (along with Nicholas Negroponte) is the One Laptop Per Child initiative[13], where Python plays a key role, and which largely revives the Dynabook idea.

COMPUTING FOR NOVICES

Despite the strong advocacy of the likes of Papert and Kay, there is no consensus on whether most people can and should learn a modest amount of programming, nor what age is best to start. Still, if we can stipulate that **some** people will become programmers, we conclude that they have to start with **some** language. Is it important which language that is?

Among the most famous quotations attributed to the great computer scientist Edsger Dijkstra are critiques of teaching computer science with inappropriate languages, notably that "The use of COBOL cripples the mind; its teaching should, therefore, be regarded as a criminal offense" and "It is practically impossible to teach good programming to students that have had a prior exposure to BASIC: as potential programmers they are mentally mutilated beyond hope of regeneration." Many people have taken DIjkstra's advice to heart, and designed languages intended to expose young adult students to good practice and good theory from the beginning. Pascal and Smalltalk are probably the most successful examples.

9 http://www.amazon.com/gp/blog/id/A3W4CUXPE1WFNF
10 http://dmoz.org/Computers/Programming/Languages/Lisp/Logo/
11 http://en.wikipedia.org/wiki/Logo_%28programming_language%29
12 http://www.mprove.de/diplom/gui/Kay72a.pdf
13 http://www.laptop.org/

Another effort in this direction was the ABC programming language project of Geerts, Meertens and Pemberton at Centrum voor Wiskunde en Informatica (CWI) in Holland. They list some advantages of ABC in comparison to Pascal as 1) small, intuitive syntax; 2) maps well onto design principles; 3) terseness; 4) support for refinement; 5) no declarative statements; and 6) interactive[14]. Along with these good ideas the CWI group had the inspiration to put the young Guido van Rossum on staff, and it is here that the Python story began.

While much of the history of the Python community remains an oral history, we do have the following quote from Guido: "Over six years ago, in December 1989, I was looking for a 'hobby' programming project that would keep me occupied during the week around Christmas. My office would be closed, but I had a home computer, and not much else on my hands. I decided to write an interpreter for the new scripting language I had been thinking about lately: a descendant of ABC that would appeal to Unix/C hackers[15]." This was the origin of Python - the combination of a well designed learning language and a practical scripting language.

COMPUTER PROGRAMMING FOR EVERBODY

Once Python had attained a certain momentum, Guido van Rossum under the auspices of the Corporation for National Research Initiatives applied for and received a small grant to support a project called Computer Programming for Everybody (or CP4E), to investigate democratizing programming. The project was funded for a single year, half the anticipated duration, through the Defense Advanced Research Projects Agency[16] (the same agency that funded the initial development of the internet). Guido and much of the core development team moved into the for-profit sector at about that time as well. The proposal[17] lists broad objectives of developing high school and college curriculum, developing tools, and developing a user community, as well as many subcomponents of these objectives.

The motivation section[18] is in the intellectual lineage of Papert and Kay:

> *“If we are successful, non-experts will be able use their computers and other intelligent devices much more effectively, reducing their level of frustration and increasing their productivity and work satisfaction. (New leisure possibilities will undoubtedly ensue as well!) Computer users will be able to solve their own computer problems more often, reducing the need for technical support.*
>
> *Even if most users do not program regularly, a familiarity with programming and the structure of software will make them more effective users of computers. For example, when something goes wrong, they will be able to make a better mental model of the likely failure, which will allow them to fix or work around the problem. They will also be able to assess better when they can make the changes themselves and when they will need the services of an expert. They will be more able to converse with experts, since they will now share more of a common language. An analogy is obtaining basic literacy in automotive maintenance: you know enough to check your oil and add a few quarts if necessary, but you also know that you shouldn't try to change your own brakes. When the mechanic says 'your rotors are warped and you need new pads,' you understand what he is talking about.*
>
> *If this effort is successful, there could be many millions, eventually billions of computer programmers, at various levels of proficiency. The effects this will have on the state of the art of software development is hard to imagine. The nature of software will change to accommodate the needs of these programmers, allowing customization through source code modifications--and personalizations will be plentiful.”*

14 http://homepages.cwi.nl/~steven/abc/
15 www.python.org/doc/essays/foreword/
16 http://en.wikipedia.org/wiki/DARPA
17 http://www.python.org/doc/essays/cp4e.html
18 http://www.python.org/doc/essays/cp4e.html

The funded CP4E project's principal achievement was developing the IDLE system. IDLE is a clean and lightwieght development environment in the tradition of the early Borland IDEs. Because of its simplicity and convenience, it remains the development platform of choice for beginners in many contexts.

Still, it must be admitted that many of the goals of CP4E remain unfulfilled to this day. The SIG that was centered on the project, while hosting many fascinating discussions, lost focus. Until recently, a fairly large number of fairly small projects were developed in and around the cluster of ideas that motivated CP4E and the edu-sig. Efforts to create a directory and a repository for this work have so far failed to reach critical mass. There appears to be a cat-herding problem, with each enthusiast having his or her own individual goals. That said, a great deal of fascinating work has been and continues to be happening, and quite a few young people have been exposed to Python as their first programming language and environment. Perhaps even more importantly, the mission of CP4E, and Python's role in it, has been an active goal for quite a few people in the intervening seven years.

The section “Further Information” at the end of this article attemps, and surely fails, to be an exhaustive list of Python related education projects. It includes projects where Python is or has been used to develop a domain specific learning language or learning tools, or where Python is used as the language of discourse in teaching programming, or in using programming to teach other topics. It is limited to resources aimed at novice programmers who are not necessarily going to become technical or computing professionals.

WHY PYTHON?

Pulling together these Python as a First Language/CP4E experiences, formalizing them, and making a case for them in educational settings, remains somewhat elusive. If we accept Dijkstra's admonitions that “a first computing language is important to the intellectual development of the student”, it seems to those of us who appreciate Python that it would be a good choice. However, it is arguably our responsibility to get beyond "we like it because it is good" for the non-Python audience.

The Python community knows what it likes and is amazingly verbose and articulate in discussing this or that programming construct, but we have never been particularly adept at explaining our vision to those who have not experienced it. In researching this article, I made some effort to get Python First *[Ed – the concept of Python as the first programming language to learn]* enthusiasts to explain the basis of their enthusiasm in the context of the novice programmer. I found some people who were willing to focus on certain features of the language, resembling the features of ABC described above.

I would like to venture an alternative explanation (inspired to some extent by comments from Kirby Urner and from Ian Bicking about how Python relates to object-oriented programming). That Python is arguably the best language for beginners and arguably the best main language for professionals is neither a coincidence nor a remarkable achievement. The properties that make Python uniquely powerful arise from its being codesigned as a language of learning and a language of production simultaneously and at its roots.

It's the quiet power of Python that makes it such a revelation to those of us who discover it after years of struggling with less expressive languages. I don't know if any of the core group has ever expressed what makes something Pythonic (except, perhaps, in cryptic zen terms), but I suggest it is that the power of Python is there when you need it and stays out of your way when you don't need it. Pythonic code then is code which is in keeping with this principle.

The positive consequences of this aesthetic seem to the Python devotee to be stunningly vast. Some of them reflect directly onto beginners. Consider that expert code in Python is indistinguishable from beginner code in those cases when the expert is doing something a beginner might do. The amount of unlearning required to become expert in Python is very small, and the amount of incomprehensible baggage required to start is zero. In other words, Python avoids disrespecting beginners with ugly and tedious compromises, and leaves a clear path to the study of logical and algorithmic thinking, minimally burdened by arbitrary syntax.

AUDIENCES AND OBSTACLES

In discussing the future of Python as a first language, we need to distinguish between several populations of novices:

1) a child working with a parent or other adult

2) a child working alone

3) a child studying a formal course in school

4) a college student studying a formal course

5) an adult seeking to achieve some personal software goal

These audiences have distinct constraints and distinct objectives. It seems clear that, except for very young children, for whom a limited environment such as eToys or Alice may be of great value for more than a few weeks, Python is clearly a viable choice.

Python's competitive position is weakest in the second and third categories, for quite distinct reasons.

In the case of a purely self-motivated child, the child's immediate objective is to do something cool on a web page and email the URL to friends and family. Python is, of course, a very powerful weapon in the hands of the experienced programmer, especially in combination with the vast alphabet soup of frameworks and libraries available. Unlike the core language, though, this power does not scale down to simple applications. Most mass market ISPs do not support Python CGIs, and certainly don't support the likes of Twisted or mod-Python. Also, the libraries and frameworks, though powerful, are conceptually inaccessible to the novice. To the extent that frameworks like Django are easy to use, they probably move far away from Dijkstra' concepts of a pedagogically useful language, and at the extreme Zope/Plone is more of a bureaucracy than a development platform. Compare this with the ease of use and remarkably complete and accessible documentation of PHP, or the zero barrier to entry of Javascript, and we lose the budding web program to pedagogically less valuable languages. Perhaps we needn't worry about the self-motivated young programmer, though. Through their enthusiasm they will learn the limitations of their platform, and eventually move to Python or a similar language. As a youth I programmed in uglier things than PHP and yet lived to tell the tale.

More of an issue is schooling. Adele Goldberg's PyCon keynote was largely a lament about trying to work within the US public school systems. She is giving up on it as a primary goal of her efforts, for a sad litany of reasons. Although her root causes were familiar to me (too much politics, too much fear, no room for experimentation), despite her sad tale she missed a couple of points that are relevant to adding a computing language to the curriculum that would not agitate the good Dr. Dijkstra. First of all, computing (except for advanced placement) is considered a vocational pursuit, not an intellectual one! This madness is endemic; Jeff Elkner's diligent pursual of all that is Pythonic, within a public school system in Virginia, all occurs in the context of a vocational track. Python is considered in competition with VBasic in this context. For college-bound students, Java has become the language of choice as a consequence of the structure of the Advanced Placement examinations. (Everyone seems to agree that this is a pedagogical improvement over C++, which presumably was as baffling to the teachers as the students.) The consequence is that despite the enormous scale of the school system, which would allow for rigorous experimentation and improvement, most schools are stuck teaching Java and VBasic.

In the home or the college, these constraints are not operative. Python First continues to gain momentum in these settings.

ENTER OLPC

Amid all of this mixture of success and frustration, a new five hundred pound gorilla has entered the room. The One Laptop Per Child initiative has promise of reviving the original vision of CP4E:

> *The project's origins go back more than four decades to the early days of computing, when most machines were still the size of small dinosaurs, and almost no one dreamed they would ever be suitable for children. But pioneering thinkers like Seymour Papert disagreed sharply, and over time led the long march from radical theory to reality proving the immense power of the personal computer as a learning tool for children[19].*

Even better, much of the system, and specifically the operating system GUI (called Sugar)[20] will be based in Python. Some clever architecture will even allow the child to play with the code! The very large scale and funding of this initiative promise to provide new focus to the CP4E community.

OLPC does an end run around the school bureaucracies of developed countries. In less developed countries, bureaucratic and political obstacles may also be less developed. It provides a Python-friendly platform with massive distribution among children who have not seen code. It promises to enhance the prominence of Python and allow much larger scales of experimentation with pedagogic ideas, all the while being perceived as great fun.

It might not be in time to save the world, but we can hope for that too.

Further Information

OLPC and Sugar	
OLPC Interface Guidelines	http://wiki.laptop.org/go/OLPC_Human_Interface_Guidelines
OLPC / Sugar Demo	http://youtube.com/watch?v=DwzCsOFxT-U
Sugar	http://wiki.laptop.org/go/Sugar
Other Python Software	
Childsplay	http://childsplay.sourceforge.net/dev.php
Cruncy	http://code.google.com/p/crunchy/
Guido van Robot	http://gvr.sourceforge.net/
Pata Pata	http://sourceforge.net/projects/patapata
PyLogo	http://pylogo.org/
RUR-PLE	http://rur-ple.sourceforge.net/
Programming for the Very Young	
Alice	http://www.alice.org/
OLPC-Squeak Demo	http://www.youtube.com/watch?v=MShr7ZHsOfI
Squeak	http://www.squeakland.org/
Video	
Introducing Python	http://www.ibiblio.org/pub/multimedia/video/obp/IntroducingPython.mpg
Various (Especially Python-in-Mathematics	http://www.4dsolutions.net/ocn/cp4e.html

19 www.laptop.org/vision/index.shtml
20 http://wiki.laptop.org/go/Category:Sugar

Videos by Urner)	
Presentations	
The computer revolution that hasn't happened yet (Kay 1998)	http://www.educause.edu/content.asp?page_id=666&ID=COM9802&bhcp=1
Presentation to the Austin Python User Group (Tobis 2007)	http://webpages.cs.luc.edu/%7Emt/Python-First/Py1.html
Papers	
Models of Growth (Cavallo 2004)	http://www.media.mit.edu/publications/bttj/Paper11Pages96-112.pdf
Using Python in a High School Computer Science Program (Elkner 2000)	http://www.python.org/workshops/2000-01/proceedings/papers/elkner/pyYHS.html
A Personal Computer for Children of All Ages (Kay 1972)	http://www.mprove.de/diplom/gui/Kay72a.pdf
Promoting Computer Literacy through Programming Python (Miller 2004)	http://www.python.org/files/miller-dissertation.pdf
Python First: A Lab-Based Digital Introduction to Computer Science (Radenski 2006)	http://www1.chapman.edu/~radenski/research/papers/python-iticse06.pdf
CP4E (van Rossum 1999)	http://www.python.org/doc/essays/cp4e.html
Teaching Scientific Programming Using Python (Williams)	http://www.pentangle.net/python/report.pdf
Wiki's, Blogs, Articles etc.	
C2 Wiki: Computer Programming for Everybody	http://c2.com/cgi/wiki?ComputerProgrammingForEverybody
Alan Kay and OLPS (Windley 2006)	http://www.windley.com/archives/2006/02/alan_kay_the_10.shtml
Interview with Jeff Elkner (Willison 2000)	http://www.oreilly.com/pub/a/oreilly/frank/elkner_0300.html
OLPS (IEEE 2007)	http://spectrum.ieee.org/print/4985
Portals	
Python.org: edu-sig	http://www.python.org/community/sigs/current/edu-sig/
Python.org: Beginner's Resources	http://wiki.python.org/moin/BeginnersGuide/NonProgrammers
Python Bibliotheca	http://www.ibiblio.org/obp/pyBiblio/
Online Books and Courses	
An Introduction to Programming (Rollins)	
Software Carpentry (Williams)	http://osl.iu.edu/%7Elums/swc/
How to Think Like a Computer Scientist: Learning with Python (Downey, Elkner and Meyers 2004)	http://www.ibiblio.org/obp/thinkCS/python/english2e/html/ http://www.greenteapress.com/thinkpython/
The LiveWires Python Course	http://www.livewires.org.uk/python/
Python First Textbookes	

Learn to Program Using Python (Gauld 2001)	http://www.amazon.com/Learn-Program-Using-Python-Self-Starters/dp/0201709384/ref=pd_bbs_8/103-6355689-1543059?ie=UTF8&s=books&qid=1176765148&sr=8-8
Object-Oriented Programming in Python	http://vig.prenhall.com/catalog/academic/product/0,1144,0136150314,00.html
Introduction to Computing and Programming in Python, A Multimedia Approach	http://www.amazon.com/Introduction-Computing-Programming-Multimedia-Approach/dp/0131176552/ref=sr_1_1/103-6355689-1543059?ie=UTF8&s=books&qid=1176872036&sr=8-1
Beginning Python: From Novice to Professional	http://www.amazon.com/Beginning-Python-Novice-Professional/dp/159059519X/ref=pd_bbs_3/103-6355689-1543059?ie=UTF8&s=books&qid=1176765209&sr=1-3
Python Programming: An Introduction to Computer Science	http://www.amazon.com/Python-Programming-Introduction-Computer-Science/dp/1887902996/ref=sr_1_8/103-6355689-1543059?ie=UTF8&s=books&qid=1176765209&sr=1-8

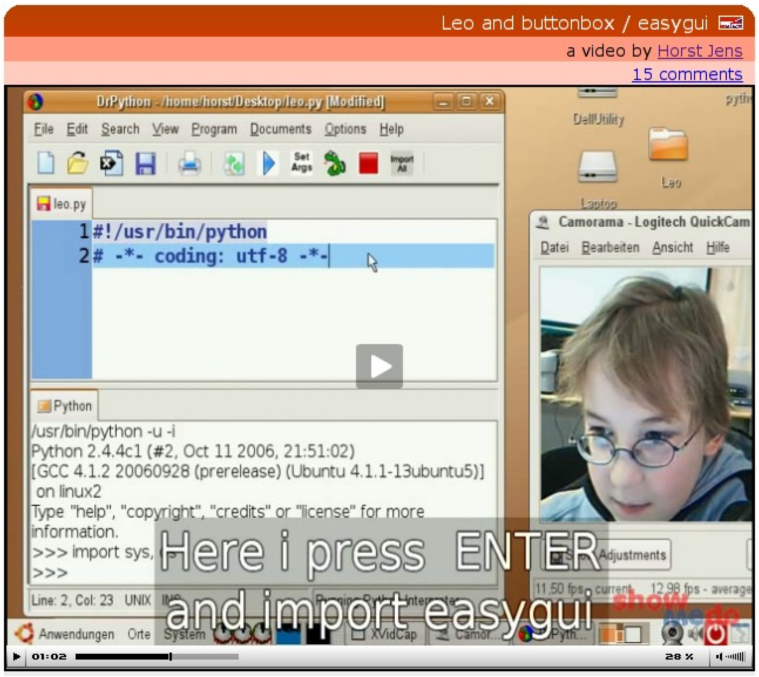
Python for Kids

Tennessee Leeuwenburg

Horst Jens, one of our readers, wrote in to let us know about a great project he is involved with: teaching Python to primary school students in Austria.

The videos themselves are available from the ShowMeDo website[21], which is a site that accepts user-submitted videos on a variety of software topics. There is an area dedicated to Python, but other languages are also covered.

The ShowMeDo site also allows for users to create wiki spaces for their content. Horst Jens has set up just such an area for his students[22], including a description of each students' presentation, the technology they are using and the source code for their work.


Leo and buttonbox / easygui
a video by Horst Jens
15 comments
DrPython - /home/horst/Desktop/leo.py [Modified]
File Edit Search View Program Documents Options Help
leo.py
Python
Camorama - Logitech QuickCam
Here i press ENTER
and import easygui
01:02
28 %

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
```

```
/usr/bin/python -u -i
Python 2.4.4c1 (#2, Oct 11 2006, 21:51:02)
[GCC 4.1.2 20060928 (prerelease) (Ubuntu 4.1.1-13ubuntu5)]
 on linux2
Type "help", "copyright", "credits" or "license" for more
information.
>>> import sys,
>>>
```

Line: 2, Col: 23 UNIX

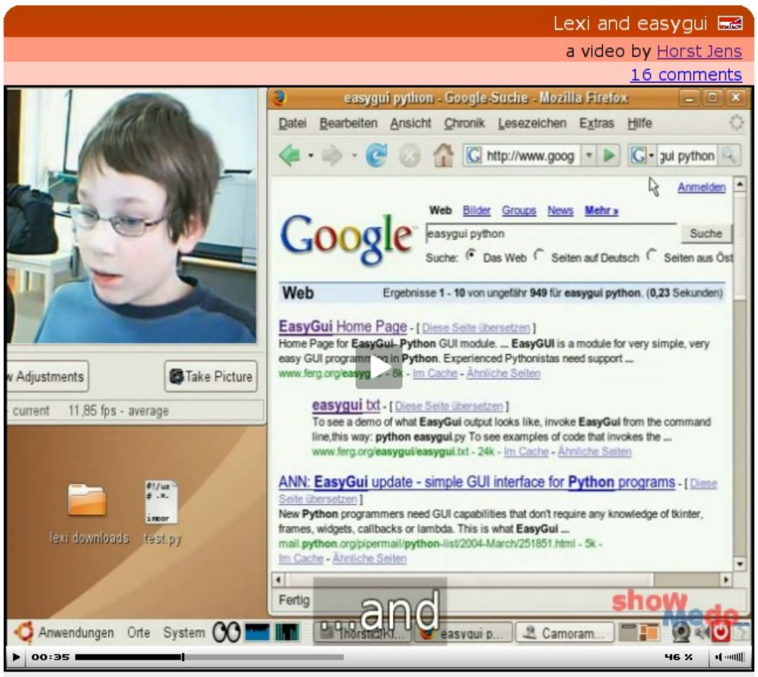
Leo demonstrating PyDoc and EasyGUI

The first student who we cover here is Leo, who demonstrates how to use PyDoc to understand the buttonbox function of EasyGui. The screenshot shows him using DrPython[23] to edit his code. The presentation includes a video of Leo (on the right of the screen) as he works, and a screencast (on the left) as he types and works. English subtitles have been added in order to be accessible to an international audience.

Next we take a look at Lexi's presentation. Lexi shows us how to download a third-party module.

Downloading a third-party module is something which most developers will have to tackle and is done differently in each language. Since many important functions are often provided by third-party modules, it is important to be comfortable with their installation and use.

In his presentation, Lexi downloads the EasyGui module. He then performs a few tasks using DrPython to show that EasyGui has been installed successfully and how it may be used.


Lexi and easygui
a video by Horst Jens
16 comments
easygui python - Google-Suche - Mozilla Firefox
Datei Bearbeiten Ansicht Chronik Lesezeichen Extras Hilfe
Google
Web Bilder Groups News Mehr »
Suche
Adjustments
Take Picture
lexi downloads
test.py
Fertig
and
00:35
46 %

easygui python

Suche: Das Web Seiten auf Deutsch Seiten aus Öst

Web Ergebnisse 1 - 10 von ungefähr 949 für easygui python. (0,23 Sekunden)

EasyGui Home Page - [Diese Seite übersetzen]
Home Page for EasyGui- Python GUI module. ... EasyGUI is a module for very simple, very easy GUI programming in Python. Experienced Pythonistas need support ...
www.ferg.org/easyg... - 8k - Im Cache - Ähnliche Seiten

easygui txt - [Diese Seite übersetzen]
To see a demo of what EasyGui output looks like, invoke EasyGui from the command line,this way: python easygui.py To see examples of code that invokes the ...
www.ferg.org/easygui/easygui.txt - 24k - Im Cache - Ähnliche Seiten

ANN: EasyGui update - simple GUI interface for Python programs - [Diese Seite übersetzen]
New Python programmers need GUI capabilities that don't require any knowledge of tkinter, frames, widgets, callbacks or lambda. This is what EasyGui ...
mail.python.org/pipermail/python-list/2004-March/251851.html - 5k -
Im Cache - Ähnliche Seiten

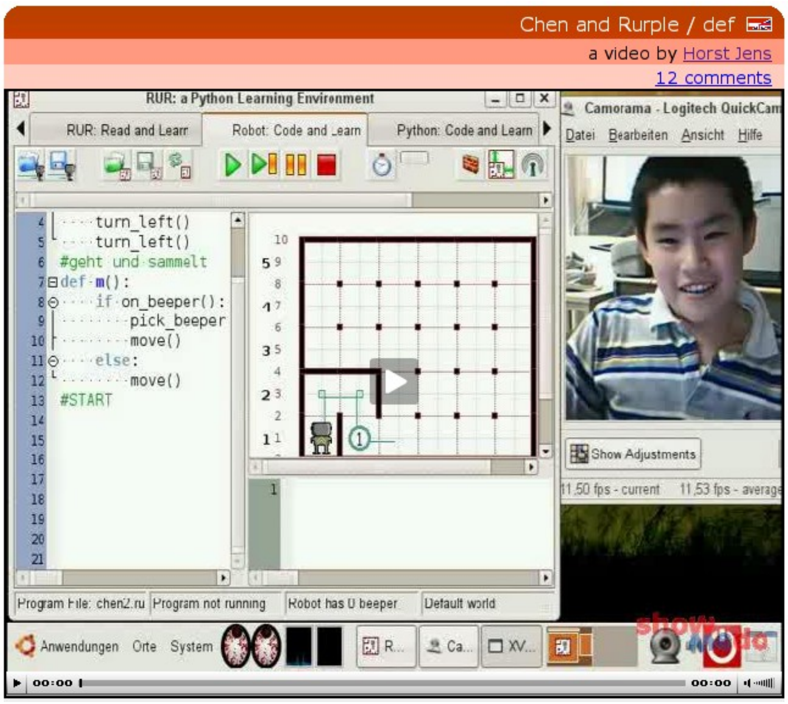
Lexi demonstrating a third-party module

21 http://showmedo.com/videos/series?name=pythonJensFromKidsSeries
22 http://wiki.showmedo.com/index.php/PythonJensFromKidsSeries
23 http://drpython.sourceforge.net/

The next presentation is by Chen, who shows how to define a function using Rurple.

As the wiki puts it, 'Rurple is a very cool, pretty and multi-lingual learning environment for python. It features a little robot that can only be moved forward and made [*sic*] a 90° left-turn, but he understands some python commands! Using Rurple is a great way to learn/teach/use python'.

These presentations show how Python can be used as a great introductory tool for learning about computers and programming for the first time. The video and screencast media really show how familiar the students are with what they are doing.



```
    turn_left()
    turn_left()
#geht und sammelt
def m():
    if on_beeper():
        pick_beeper
        move()
    else:
        move()
#START
```

Chen demonstrating Rurple

It would be great if *Python Papers* readers could visit the ShowMeDo website and leave their comments.

The use of Python in education is clearly a significant topic in its own right. In the previous article, Michael Tobis presented an excellent overview of Python in education. As can be seen, Leo, Lexi and Chen have made some fantastic videos. If *Python Papers* readers know of any other examples of Python being used in education, or would like to let us know their views on this topic, we would love to hear from you!

PyCon 2007 Report

Brett Cannon – *amended from his blog post of 2 March*[24]*.*

I am now back in Vancouver and so I can finally take a breather, sit down, and write about PyCon 2007.

I should begin with the statement that the trip was fantastic! It was probably the best PyCon for me personally as well as the best one in terms of content (or so I heard; more details in a bit). It was also the largest with 594 people in attendance which is amazing considering that is a 43% growth rate compared to last year!

My trip began with an early flight out of YVR (Vancouver) to DFW (Dallas / Fort Worth). When I landed I got my bags, found Michael Hudson, and then met up with Richard Jones (thank you, SMS) to take a taxi from the airport to the hotel. It was a handy thing to plan hotel rides with both Michael and Richard ahead of time so as to not have to lug my tired bum to the airport all by my lonesome.

Once there we dropped off our bags and started socializing (with the attendance so loud you could be guaranteed to run into some PyCon attendee at the bar in the hotel). Initially Richard, David Goodger, one other guy (I suck with names but I am great with faces, sorry), and I went to Wal-Mart to get bowls, spoons, milk and cereal so that we didn't have to pay ridiculous prices for breakfast everyday. (David came along to guide us to Wal-Mart as he

Illustration 2: Richard Jones (Roundup) attempting to take a photo of himself and the author, not knowing the author is known for ruining photos. (Copyright Richard Jones, 2007)

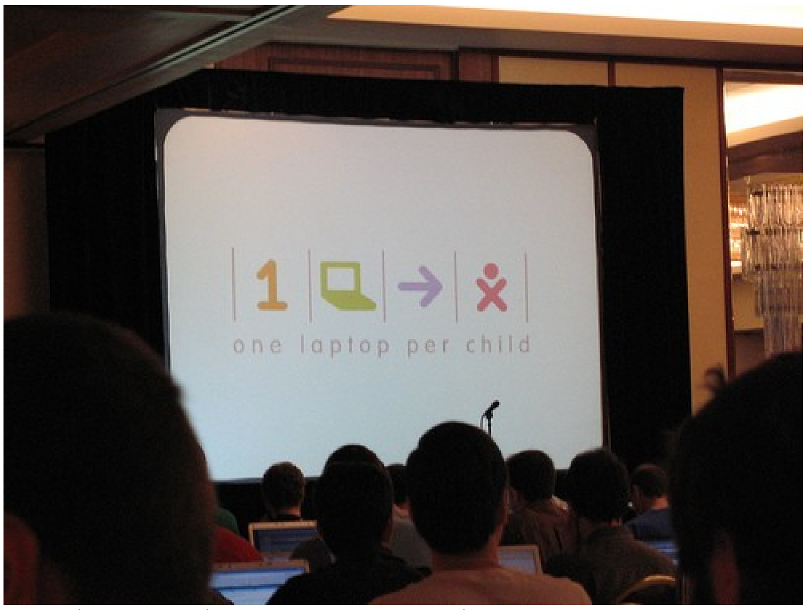
24 http://sayspy.blogspot.com/2007/03/pycon-2007-report.html

went the day before for his breakfast supplies).

When we got back, we sat in the bar chatting with various people as they walked past. After people had a drink we went to BJ's for dinner (along the way learning that birds in Texas like to congregate together in trees and let their bowels relax when they get slightly excited when you walk underneath them). I ended up sitting with Richard, Thomas Wouters, Guido van Rossum, Barry Warsaw, and Jeremy Hylton. After eating, Richard left because he was exhausted (his flight is like 26 hours or something from Melbourne). The rest of us chatted for a while and then eventually went back. I don't remember exactly what I did for the rest of that night.

One note about all the meals I ate: I pretty much mention every dinner I ate during the conference as it helps me remember what I did. Most nights it was not that interesting, nor do I expect people to care where I ate on some night, but it's my essay so you just have to live with the unnecessary details.

Friday was the beginning of the conference. Ivan Krstic's keynote was great! I know that OLPC has picked up a lot more developers because of Ivan's presentation. He also gave out rev-1 hardware to one guy who recognized what one math equation did and one to Guido during his keynote (several other Python luminaries ended up with laptops throughout the conference; alas I was not one of them). It was entertaining to watch people want to touch those things for the rest of the conference. I think it gave Guido a nice break from being the center of attention as he usually is at PyCon.



The intro slide to Ivan Krstic's day one keynote on the One Laptop Per Child (OLPC) project.

After the keynote I did what I did last year: I ignored almost all talks and hacked. I decided I wanted to get my PEP 362 implementation finished before the sprints started (and I did). It was interesting developing some code that is both 2.6 and 3.0 compatible. The biggest trick was remembering to use 'print' the statement or the function.

One of the talks that I did attend on Friday was the python-dev panel that I organized. Steve Holden did a good job as moderator. We had Raymond Hettinger, Neal Norwitz, Andrew Kuchling, Jeremy Hylton and myself on the panel. Thomas Wouters and Barry Warsaw answered some questions from the audience. People seemed to really appreciate the panel (even though the panelists pretty much all wished they could also hear the PyPy taking place next door). Thanks to the panel, it looks like Py3K development will move over to a distributed version control system (most likely Bazaar), in order to make merging back into the trunk easier by having an individual branch for each feature (thus make managing them a simpler action).

I also attended both lightning talks on Friday. I heard people question whether Mike Orr and Jacob Kaplan-Moss kept people to five minutes (which they did) because some of them seemed rather long; that is just a side-effect of boring lightning talks. They were still entertaining overall, though. I gave a lightning talk in the evening about removing automatic unpacking for tuple parameters. The talk, though, backfired as people misread ``def fxn(a, (b, c), d): pass`` as ``fxn(a, *(b, c), d)`` somehow. People were fairly confused throughout most of the lightning talk which basically killed it. Oh well. Lesson learned about trying to quickly ask a large audience about an obscure piece of syntax. In the end, it didn't matter as this feature of Python has been approved for removal since PyCon ended.

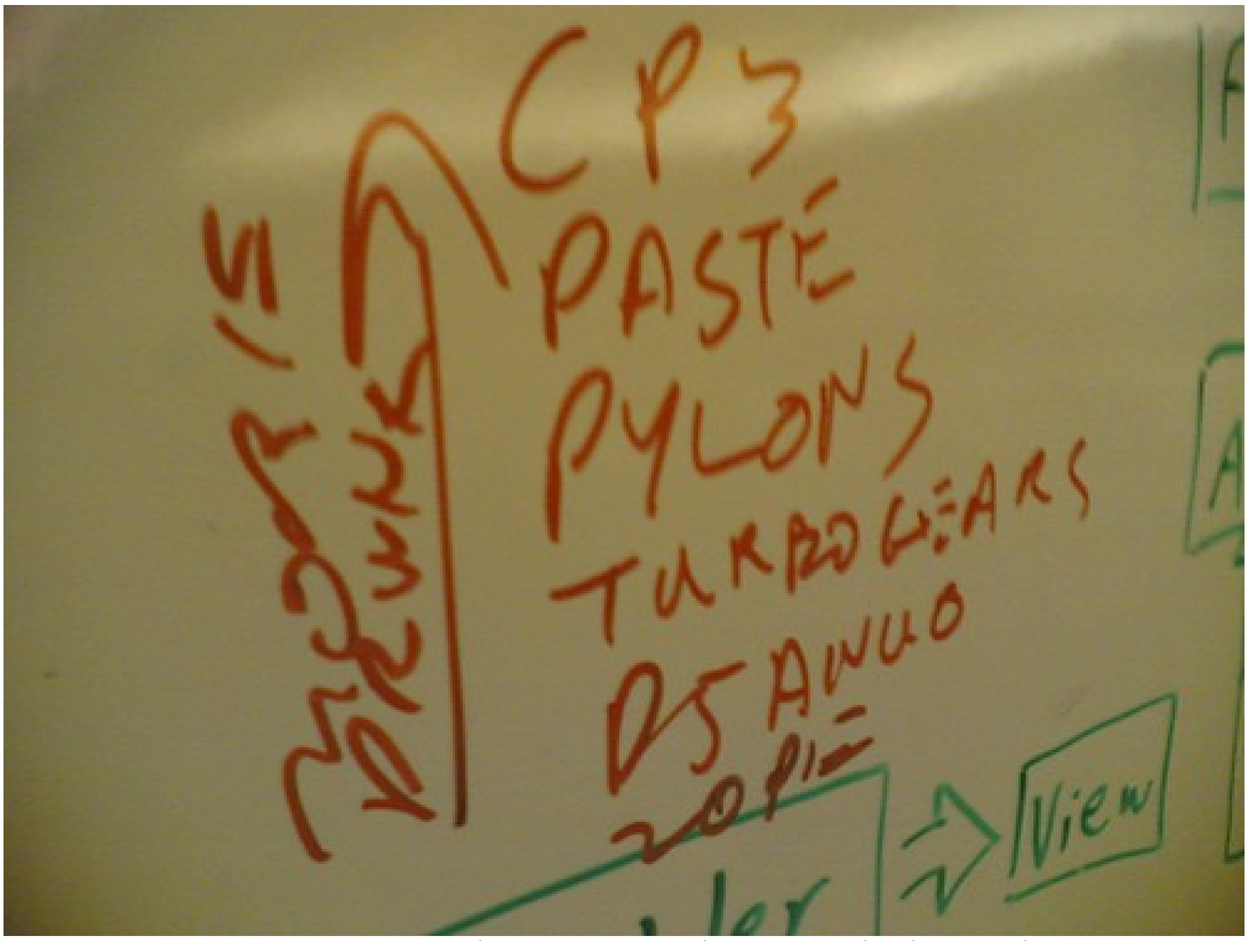


Photo by Jacob Kaplan-Moss (Django) of a whiteboard listing various frameworks and the order of preference.

I then hacked on PEP 362 stuff until the Python Software Foundation meeting Friday night. The thing went for three hours! At least we got a free dinner out of it. I managed to get re-elected to the board of directors and "promoted" to executive vice president. I also stayed on as chairman of the infrastructure committee to see through the eventual transition to the new issue tracker.

After the meeting there was plenty of partying on the 12th floor of the hotel in the EWT suite. Raymond Hettinger and Martin Blais both work there along with Christian Tismer doing consulting work. Throughout the conference, that hotel room tended to end up being the place to go at night when you wanted a beer from their bathtub, some wine, or to play some pool.

Saturday's keynote by Adele Goldberg was ignored by me for preference to PEP 362 coding (my mother is a teacher so the whole educational thing is nothing new to me). I did attend the "Scaling Python for High-Load Web Sites" but only partially paid attention. When I attended Jim Hugunin's IronPython talk, though, I listened. Jim's talks are always fun and informative. He has done a great job with recognizing that not everything Microsoft does is great but still giving proper credit when they do things right; in other words he is fair and balanced in terms of his views on Microsoft stuff. Plus Jim always does something cool in his talk, this year being the use of Python to control a little robot that grabbed a ball on the floor using MS's robotics studio (who seeked him out to get Python support!). He also showed a game he coded up using MS's XNA platform (which unfortunately cannot run on the XBox 360 yet as their version of .NET lacks reflection support).

Guido gave his Py3K keynote on Saturday during lunch. People who attended seemed to come out of it much more calm and happy about Py3K and the direction it is taking. Various other people also said they were realizing that the transision from 2.6 to Py3K was being taken seriously and would be as painless as possible. That was a critical event at PyCon as previously many people were panicking about moving their code from the 2.x series to 3.0, but those worries seem to have been calmed.

Saturday also included my security talk. People said it was a great talk and it seemed well-received. No one pointed out a flaw in my design which was good. Overall, people were very supportive and wanted me to get it done and included in the core, which is planned.

That night I went with Thomas, Andy (an officemate of Guido's), Neal, Guido, Jeremy, Barry, and the Python editor from O'Reilly to an Ethiopian restaurant. I learned that Barry is allergic to a bunch of stuff that night.

Sunday's keynote by Robert Lefkowitz was good, but once again I only partially paid attention because of PEP 362 coding. (I basically rewrote the thing from scratch over the span of the conference, if you had not picked up on that fact based on the amount of time I spent on it.)

In terms of talks I went to Sean Reifschnedier's vim talk (Sean is also the man who made the wireless stable this year so kudos to him for that) and the Pybots talk. I discussed some things with Grig Gheroghiu and Titus Brown about testing stuff which was positive.

I saw the lightning talks at the end of the day. The highlight was seeing Richard's Pyglet talk since it had pretty pictures.

At the end of the conference it was announced that PyCon 2008 will be in Chicago, IL. That should be a lot of fun. I went once to Chicago and I loved it. We also did sprint introductions where I represented the core like I seem to do every year. Managed to get some people to join us on the core sprint (probably one of our largest groups in terms of people sticking around for most of the sprint which was great).

For dinner a bunch of us went to a really nice French restaurant that had the chef from the same expensive restaurant we went to last year (The Standard, which was the fancy restaurant we went to last year, had closed a few weeks prior).

Monday was the first day of sprinting. We agreed to the distributed VCS usage for Py3K (Thomas is in charge of making it happen which will probably be in a couple of months). Thomas also backported dict views to 2.6, which is in a branch currently. He also merged his slice removal branch in Py3K (this does not mean that slicing is not possible, it is underlying slicing stuff at the C level).

Initially Pete Shinners (of PyGame fame) did work on adding methods to the bytes type. He eventually moved over to helping Patrick and Eric on a C implementation of PEP 3101 (Advanced String Formatting).

Jerry (whose last name I don't know) converted a bunch of old-style tests to use unittest. Later on his friend Mike helped out.

But first Mike (whose last name I don't know), Daniel (someone else whose last name I don't know), Guido, and I discussed the new I/O design for Py3K. Mike and Daniel then wrote up the PEP which has subsequently been checked in.

Jeremy squashed some crashers that Armin Rigo found (read: a real pain to fix as Armin is notorious for finding really round-about ways of crashing the interpreter).

Neal was supposed to be getting xrange() to become range(), but he got distracted by removing the exceptions module since it serves no real purpose thanks to the module's entire contents being in the built-in namespace.

Guido did stuff with the new I/O library mostly. He also rewrote reload in pure Python.

Barry and I figured out a mechanism to handle the renaming of modules. Using an importer/loader for sys.path and sys.meta_path we came up with a way to map old module names to new ones, basically making the old names virtual.

Monday for me was spent hacking on cleaning up exceptions in Py3K. I made it a TypeError to raise or catch objects that did not inherit from BaseException. I also removed the ability to index/slice on exceptions.

I went out for sushi at a bar on Monday which just seems odd; having plasma screens around you while you eat sushi just doesn't seem quite right to me.

Tuesday was more hacking and the last day Jeremy, Neal and Guido were around. I began my painful quest to remove the 'args' attribute from exceptions and make BaseException take a single argument.

Illustration 3: Day three of sprinting. On the left: Phil Hassey (Galcon), Mike Verdone, Jerry Seutter, and someone whose name escapes me. On the right: the author, Danil Stutzbach, and Pete Shinners (PyGame). Richard Jones (Roundup) is taking the photo. (Copyright Richard Jones 2007)

I ended up catching dinner with Jerry at TGIF's restaurant on Tuesday.

Wednesday night I finally finished my exception work. I have subsequently posted on python-3000 about whether people want to move forward with this as it was extremely painful to pull off (mostly from the change in BaseException's constructor only taking a single argument). It looks like the 'args' attribute on exceptions will stay and the 'message' attribute introduced in 2.5 will be deprecated in 2.6 (might be a record in terms of the lifespan of a feature before it is deprecated).

Because so many people left after Tuesday, we had some of the video game guys move over to our table, including Richard. It was rather cool having the creators of PyGame, pyOpenGL, and Pyglet all in the same room.

We all went out to Thai on Wednesday night.

Thursday I was still mentally recovering from the previous two days of frustration so I just relaxed. I also got everyone at my table hooked on Ohloh. Since a bunch of people there were creators of various projects they had fun looking at the stats on their projects. And writing this paragraph led me to spend way too much time aliasing old SourceForge names to the current svn names, mostly so I could see how I stacked up in terms of commit totals.

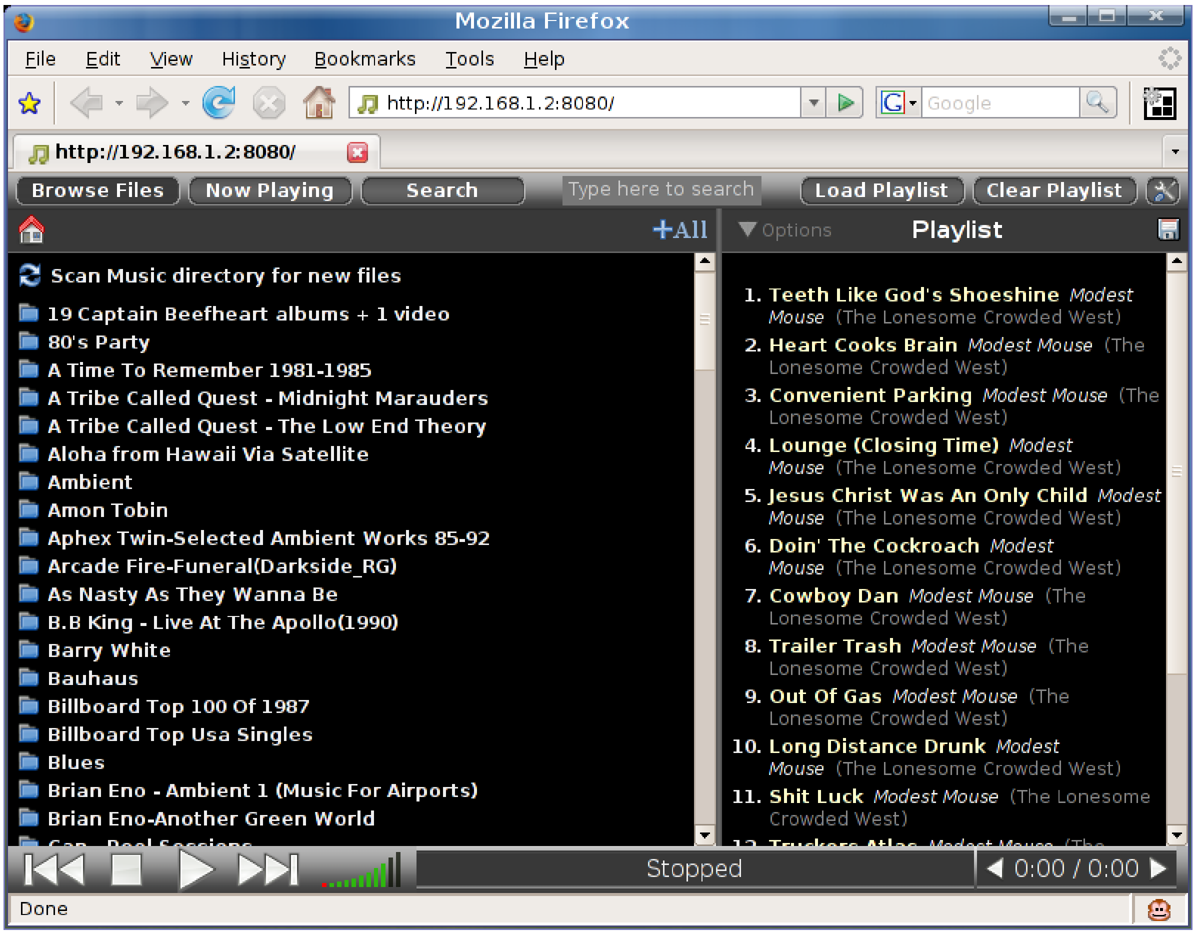
I am now home and glad to be back. The trip was great but I was exhausted by the end of it. It was great to meet so many new people (some of whom are not directly mentioned above like Michael Foord, etc.) and old friends (like David Goodger, etc.). As I said, I am already looking forward to the next PyCon as they seem to only become more and more fun!

MPD WebAMP

Chris Seickel

This rather lengthy string of acronyms stands for **M**usic **P**layer **D**aemon **Web**-based **A**synchronous **M**usic **P**layer client, and is my vision of the perfect music player.

The name pretty much says it all. MPD WebAMP provides a browser based interface to control music playback via MPD. It includes it's own CherryPy web server and provides a single page application with the features and interface one would expect from a standard desktop application. Although it is not unique in it's use of AJAX or it's use of a python based web server, it is unique in that it is a *completely* asynchronous interface which looks and behaves like an application rather than a web interface.



Scan Music directory for new files
19 Captain Beefheart albums + 1 video
80's Party
A Time To Remember 1981-1985
A Tribe Called Quest - Midnight Marauders
A Tribe Called Quest - The Low End Theory
Aloha from Hawaii Via Satellite
Ambient
Amon Tobin
Aphex Twin-Selected Ambient Works 85-92
Arcade Fire-Funeral(Darkside_RG)
As Nasty As They Wanna Be
B.B King - Live At The Apollo(1990)
Barry White
Bauhaus
Billboard Top 100 Of 1987
Billboard Top Usa Singles
Blues
Brian Eno - Ambient 1 (Music For Airports)
Brian Eno-Another Green World
Can - Reel Sessions

1. **Teeth Like God's Shoeshine** *Modest Mouse* (The Lonesome Crowded West)
2. **Heart Cooks Brain** *Modest Mouse* (The Lonesome Crowded West)
3. **Convenient Parking** *Modest Mouse* (The Lonesome Crowded West)
4. **Lounge (Closing Time)** *Modest Mouse* (The Lonesome Crowded West)
5. **Jesus Christ Was An Only Child** *Modest Mouse* (The Lonesome Crowded West)
6. **Doin' The Cockroach** *Modest Mouse* (The Lonesome Crowded West)
7. **Cowboy Dan** *Modest Mouse* (The Lonesome Crowded West)
8. **Trailer Trash** *Modest Mouse* (The Lonesome Crowded West)
9. **Out Of Gas** *Modest Mouse* (The Lonesome Crowded West)
10. **Long Distance Drunk** *Modest Mouse* (The Lonesome Crowded West)
11. **Shit Luck** *Modest Mouse* (The Lonesome Crowded West)
12. **Truckers Atlas** *Modest Mouse* (The

About Myself

My name is Chris Seickel. I am an amateur programmer who is currently managing a package on the Python Cheese Shop called MPD WebAMP. Professionally, I provide Workstation Support at a small organization and occasionally create Access databases or extend other Microsoft Office products using Visual Basic. Prior to switching full time to Linux, I found Visual Basic easy to use and thought it was the easiest language to learn and the fastest way to get things done. At that point, the only languages I had been exposed to were those in Microsoft's Visual Studio.

Eventually I tried out linux. I was immediately impressed by KDE and the amount of control I had over every aspect of my desktop. Within days I discovered Super Karamba and immediately had to create my own theme. I was amazed by how easily I picked up the scripting language that Super Karamba used. The syntax was straightforward, I typed less, had less typos and could even understand other people's code quickly. The language used by Super Karamba was Python. Being someone that is rarely content with what's available, I have been exposed to even more languages over the last few years. My initial impression that anything beyond Visual Basic would not be worth the effort to learn has been disproven, and I've learned just how much I was missing out on all of this time.

The Road to 1.0

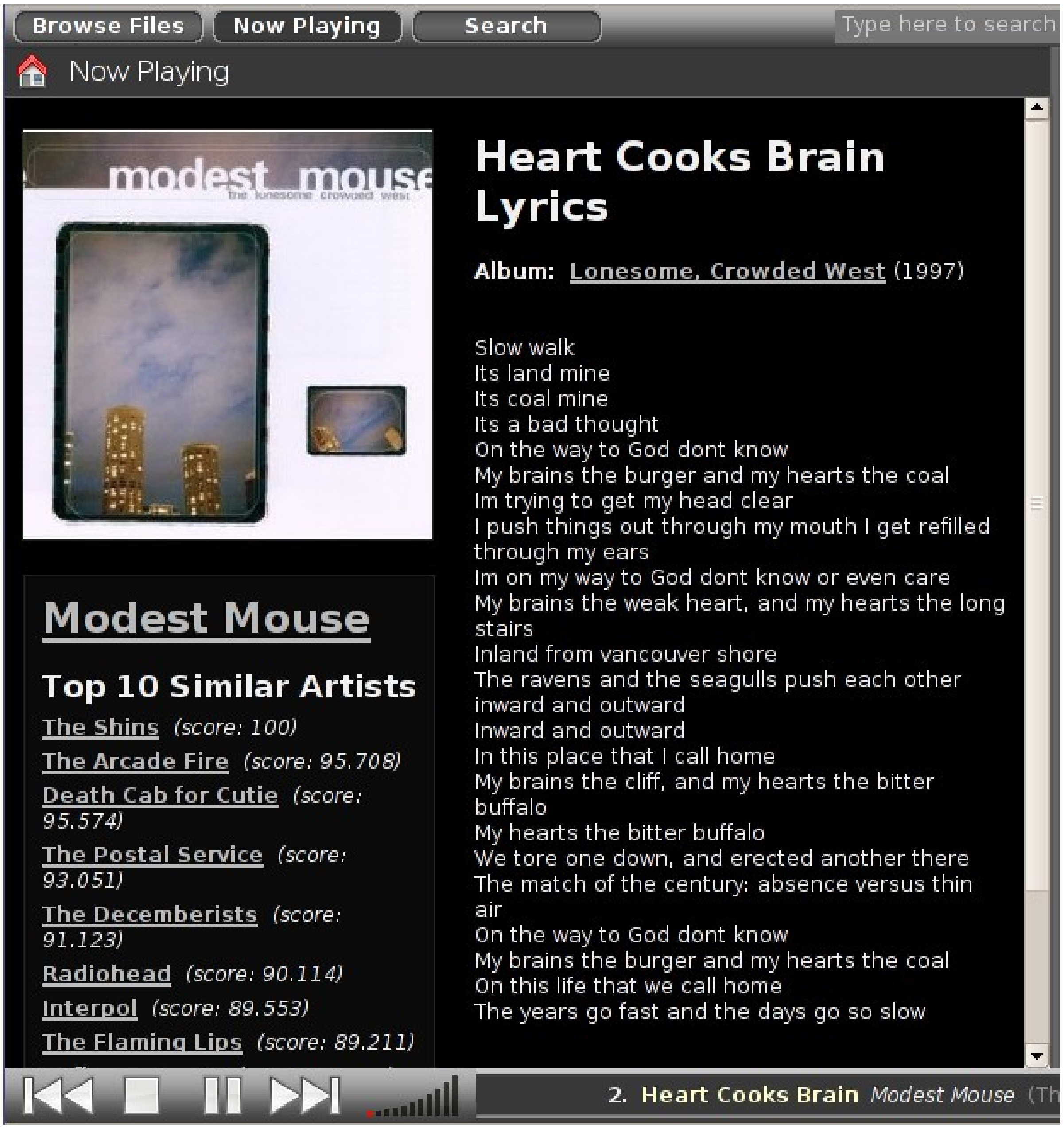
For some time I have wanted to have my PC connected to my main stereo system, as I now have more music on my computer than anywhere else. I looked at many options, such as plugging in a portable mp3 device, commercial streaming systems, or an open source system running MythTV, Freevo or Geekbox. The problem is, all of the commercially available systems would involve a loss in sound quality and require us to use a small screen interface for browsing our collection and managing our playlist. The linux media packages would allow me better sound quality, but would have the same limitations in the interface as they are designed primarily for TV screens. I also did not want to invest the time needed to set these up properly.

My wife and I are laptop junkies, and we also own a Pepper Pad. The Pepper Pad is a small tablet PC with a touchscreen based on linux which essentially provides a highly modified Firefox browser as the only graphical interface. I had heard of the Music Player Daemon for linux and decided this would fit my needs perfectly and be quick to set up. MPD is a music player that is set up as a network service, allowing you to control the same player from any computer on the network and from any of the many available clients. Since my target platforms all have one thing in common, Firefox, I decided a web interface made the most sense.

There are, of course, many web clients already available. I started experimenting with these, but then my wife asked "why can't I just use Windows Media Player?". The bar was set and the gauntlet thrown down. I had to find, or make, a web based interface to MPD that would compete with Windows Media Player. The more modifications to existing web interfaces I made, the more I learned about web programming. The more I learned, the more I realized what was possible. Since we do not use Internet Explorer at all in my house, even on the Windows PC, this opened up a whole world of technologies that no one else had explored. I quickly realized that I can mimic the layout and feel of my wife's favorite Windows Media Player in a web page using a healthy dose of CSS2, javascript, and server side scripting. The fact that I could also do this without using Flash or Java plugins intrigued me, and I felt that someone had to do it, if for no other reason than to prove that it can be done. My experiences with these plugins on 64 bit and ARM based processors left me a little annoyed at the fact that I am forced to use them to browse the web.

OK, the argument about MythTV and other systems being difficult to setup is now moot, as writing my own interface will certainly require more effort, but at this point it doesn't matter. I was actually looking for a chance to learn about web programming and I was immediately excited about this particular project. I am a geek, I can't deny it.

Unfortunately, no one else at this time had created an interface anywhere near the level of a desktop application, so I started from scratch. After a month with PHP and Dojo, I had only the most basic functions complete, having spent half my time hunting typos in PHP and digging through source code in Dojo. I started to yearn for a better server side scripting language and a faster, better documented Javascript library. That's when I found CherryPy and MochiKit, which quickly led to TurboGears, which integrates these and other Python pieces into one web development package. Within a month of switching to TurboGears I had a functional application. I could not believe how quick and easy every aspect of TurboGears was.



Heart Cooks Brain Lyrics

Album: Lonesome, Crowded West (1997)

Slow walk
Its land mine
Its coal mine
Its a bad thought
On the way to God dont know
My brains the burger and my hearts the coal
Im trying to get my head clear
I push things out through my mouth I get refilled through my ears
Im on my way to God dont know or even care
My brains the weak heart, and my hearts the long stairs
Inland from vancouver shore
The ravens and the seagulls push each other inward and outward
Inward and outward
In this place that I call home
My brains the cliff, and my hearts the bitter buffalo
My hearts the bitter buffalo
We tore one down, and erected another there
The match of the century: absence versus thin air
On the way to God dont know
My brains the burger and my hearts the coal
On this life that we call home
The years go fast and the days go so slow

Modest Mouse

Top 10 Similar Artists

The Shins (score: 100)
The Arcade Fire (score: 95.708)
Death Cab for Cutie (score: 95.574)
The Postal Service (score: 93.051)
The Decemberists (score: 91.123)
Radiohead (score: 90.114)
Interpol (score: 89.553)
The Flaming Lips (score: 89.211)

2. Heart Cooks Brain *Modest Mouse*

Now, of course, functional is not complete. There were another four months of additions, improvements and improvements on the improvements before 1.0. Along the way I got the opportunity to really delve into the languages being used and explore new ideas that I had never used before. Eventually, I felt that I was far enough along to release it as a python egg on the Cheese Shop. Almost immediately, I was contacted by another developer who had also just released an MPD web client based on a python web server, which is called WyMyPy. While our projects are different enough that we could not share any code, we at least both got to see someone else's ideas on how to accomplish the same tasks. Realizing that there are actually other people out there who may have use for the code I am writing made me look at things in a different light. I decided to try and make the project more modular so that pieces of it can be reused by other developers.

It was at that point that I finally broke down and wrote my own class for the first time. I had stayed away from classes for all these years because I never quite got why I would need them or how to construct them properly. After several hours of digging into the *Dive Into Python*[25] chapters on classes, and constructing my own class for working with lyrics searching and parsing, I finally got it. I don't want to call it a life changing event, but it was a significant event in my education in programming. All of a sudden I saw my whole project in a new light. I started to reconsider all of the tangled webs I wove to make all the pieces fit together...

At this point, I have finally made it to 1.0 and achieved all of the goals I originally set out to accomplish. There have now been about 70 downloads of MPD WebAMP 1.0.7 from the Cheese Shop and every day a few more people download it. I am happy with the result, as is my wife. We both use it everyday – it performs well and is very stable. One thing that I didn't anticipate was that being able to have multiple copies of the same music player running on different PCs has made the experience of listening to music a little more of a social event in my house. Instead of one person putting together a playlist and being in control of the music, we both add to the playlist as things pop into our head, skip tracks we don't like, adjust the volume, discuss the Now Playing information, etc.

Of course, as soon as I finish anything, I immediately start to think about how I can do it better. In this case, I actually have good reason to do so. After reaching my initial design goals, I have found that there is no room to expand within the current structure. It's time to put my new found love of classes and Object Oriented programming to good use.

What's Next?

One of the biggest problems that emerged in this project was managing the level of communication necessary to maintain a reasonable response time. The difficulty comes from the fact that you cannot send commands to the web browser that it did not request, so it is up to the client to know to ask the correct questions at the correct time. Beyond knowing what questions to ask, the client needs to deal with situations where there is more data available than it can process within a reasonable time frame. I define reasonable time frame as less than a second, as basic interface events are not processed while javascript is being executed, making the application appear "frozen".

This issue is most noticeable while downloading large playlists. The status request is a request/callback in a constant loop where the basic status of MPD is checked: state (playing, stopped or paused), currently playing song information, volume, time elapsed, total length of current song and playlist version number. To maintain a smoothly flowing time counter, this needs to be checked at least five times a second. This polling was initially managed by setting up an interval timer that requests updates every 200 milliseconds. By itself it worked great, but when you add a large playlist download, things get messy. It was common for playlist loads to take multiple seconds or longer, which caused the status updates to back up. When this happens, everything freezes until the playlist finishes loading, followed by a rapid succession of status updates that may be very old or even out of order, extending the period of unresponsiveness. This was dealt with by making the playlist request cancel the status requests, and then break every so often to send a single request, finally restarting the loop when it is done loading. This behaviour extends to other requests, and adds unnecessary complexity and ample opportunity for bugs that are difficult to recreate. What's worse is that four out of every five status requests return no new information anyhow.

The other side of the coin is that for every change to the status or playlist, the work of compiling the information to be sent is repeated for every client that is connected. In the end, both the server and the client end up wasting more time on redundant actions than they actually spend on useful activities. I've recognized for a long time that the whole process would be easier if I could simulate pushing information to the browser when the server deems it necessary, and to also find a way to prioritize updates to allow for "backgrounding" of certain downloads or allow certain updates to "jump to the front of the line".

25 http://www.diveintopython.org/

While the standard methods of request/callbacks is ideal for client initiated AJAX requests, when most of the updates will be generated by external events it becomes a different story...

Very quickly after I realized the usefulness of creating my own classes, I began to see how this new skill could be applied to fix these issues with asynchronous communications. I started to piece together an object to manage this in a whole new way that can solve all of these problems. Rather than each feature having it's own requests and callbacks and trying to make them be considerate of other requests and callbacks that may going on, I will have a single callback function in a continuous loop. This single request callback will then be used to run arbitrary code that is sent by the server. The key to this on the client side is taking advantage of the fact that javascript code does not have to be byte compiled when the page is loaded. Although it is faster when loaded this way, when all you want to do is run a one liner or call a previously defined function, you can simply use eval() to run code that is sent as a string.

Now that I have a way to send arbitrary code to the clients, I also need a way to send the correct commands to the correct clients. To accomplish this task, each client will be given their own queue, which will be stored as a dictionary containing three separate lists: high, normal and low priority. On each iteration of the polling loop, the server will send another command from the list, starting with high priority items and working it's way down to low only when the high and then normal priority lists have been emptied. In this way I can emulate backgrounding of actions and ensure that large downloads of data don't interfere with status updates. The beauty of this is that I no longer need to think about what else may be going on when sending data and commands, I only need to specify a priority and let the simple priority system sort out the details.

Now that I've solved the problem of what commands are sent when, I realized I had created the problem of which clients get what commands. By removing the client side requests in favor of a server side push, the server now needs to know what the client is doing so that it does not send the wrong updates. This problem is easily solved by having the clients connect and disconnect from various "slots". The server will maintain a list of clients that are connected to each slot, and when an event is fired for a certain slot it will go through the list and add the script to the queue of each client in that list. When a client switches from browsing files to Now Playing info, they will disconnect from the browse slot and connect to the info slot. Now, when the player moves to a new song, the server will enqueue the appropriate command to all clients connected to the status slot to update the current track and time information, as well as sending the new lyrics, album art, and similar artists info to all clients connected to the info slot.

Prioritizing is handled by maintaining three different queues for each client: High, Normal, and Low. When the client requests the next item in the queue, it will send all High priority items in one burst if any are present. If there are no High priority items, then it will send the oldest Normal priority item. If there are no Normal items left, then it will finally move on to the Low queue.

The basis of the system is JSController.push(script [, priority=""[, slot=""[, client=""]]]). A simple example of how this can be used is to update the DOM from the webserver with: js.push("$('timeElapsed').innerHTML='0:00', priority="high", slot="status"). The fact that there is now a direct connection to the DOM, via server side python, opens up many new and interesting possibilities which I am only beginning to explore.

The most common use will be using the NodeLink class as a shortcut for JSController.push():

```
from JSObject import *
js = JSController(slots=dict(root=[], status=[], info=[], playlist=[],
browse=[]))
timeElapsed = NodeLink(js, "timeElapsed", slot="status", priority="high")
```

Now, each time an attribute of timeElapsed is set, the changes will be pushed to all of the clients connected to the status slot. There is no need to worry about requests and callbacks, just set the attributes as if your were dealing with a DOM node directly.

For example, the following **Python** commands:

```
timeElapsed.innerHTML = "0:00"
timeElapsed.style = dict(width="100px", height="20px", top="10px", right="10px")
```

Will run the following JavaScript code:
(NOTE: setNodeAttributes() is a MochiKit convenience function that will be used for all attribute changes other than innerHTML)

```
$('timeElapsed').innerHTML='0:00';
setNodeAttributes('timeElapsed', 'style', {'width':'100px', 'height':'20px',
'top':'10px', 'right':'10px'});
```

See below for the actual module as it stands now:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#
#       JSObject.py
#
#       Copyright 2007 Chris Seickel <cseickel@gmail.com>
#
#       This program is free software; you can redistribute it and/or modify
#       it under the terms of the GNU General Public License as published by
#       the Free Software Foundation; either version 2 of the License, or
#       (at your option) any later version.
#
#       This program is distributed in the hope that it will be useful,
#       but WITHOUT ANY WARRANTY; without even the implied warranty of
#       MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#       GNU General Public License for more details.
#
#       You should have received a copy of the GNU General Public License
#       along with this program; if not, write to the Free Software
#       Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301,
USA.
from datetime import datetime
import simplejson
base_script = """function getUp(){loadJSONDoc('updates').addCallback(UP)}
function UP(r){
  var u=0.01
  var s=r['script']
  if(s.length>0){
    if (r['more']>0){u=0.001}
    try{eval(s)}
    catch(err){document.write(err.description)}
  }
  callLater(u, getUp)
}
getUp()
"""
class JSController:
    def __init__(self, slots=dict(root=[])):
        self.functions = []
        self.functionCount = 0
        self.slots = slots
        self.clientCount = 0
        self.queue = dict()
        self.baseScript = base_script
        self.dom = dict()
    def addClient(self):
        c = str(self.clientCount)
        self.queue[c]=dict(high=[], normal=[], low=[], count=0,
last_check=datetime.now())
        #Every client is connected to root, other slot connections may be added
seperately
        self.connect(c, slots=["root"])
        self.clientCount += 1
        return c

    def connect(self, client, slots=["root"], ):
        for slot in slots:
            if (self.slots[slot].count(client) == 0):
                self.slots[slot].append(client)

    def disconnect(self, client, slot="root"):
        self.slots[slot].remove(client)
        # Remove any pending items for this client from the slot
        for item in self.queue[client]["high"]:
            if (item["slot"]==slot):
                self.queue[client]["high"].remove(item)
        for item in self.queue[client]["normal"]:
            if (item["slot"]==slot):
                self.queue[client]["normal"].remove(item)
        for item in self.queue[client]["low"]:
            if (item["slot"]==slot):
                self.queue[client]["low"].remove(item)

    def clear(self, client):
        # Remove all pending items for this client
        for item in self.queue[client]["high"]:
            self.queue[client]["high"].remove(item)
        for item in self.queue[client]["normal"]:
            self.queue[client]["normal"].remove(item)
        for item in self.queue[client]["low"]:
            self.queue[client]["low"].remove(item)

    def push(self, script, priority="normal", slot="root", client=None):
        print "enqueing script: "+script
        if (client == None):
            clients = self.slots[slot]
        else:
            clients = [client]

        for item in clients:
            d = datetime.now() - self.queue[item]["last_check"]
            # Timeout any clients that have not checked their queues recently
            if (d.seconds > 5):
                self.disconnect(item, slot)
                print ""
                print "Removing client "+str(item)+" from slot "+slot+", timed
out."
                print ""
            else:
                # Insert item into begining, so we can use .pop() to pull the
oldest
                # item when unloading the queue
                self.queue[item][priority].insert(0, dict(script=script,
slot=slot))
```

```
                self.queue[item]["count"] += 1

    def nextInQueue(self, client):
        js = ""
        self.queue[client]["last_check"] = datetime.now()
        if (len(self.queue[client]["high"]) > 0):
            # The entire high priority queue is processed in every call
            while (len(self.queue[client]["high"]) > 0):
                s = self.queue[client]["high"].pop()
                js += s["script"]+";"
                self.queue[client]["count"] -= 1
        elif (len(self.queue[client]["normal"]) > 0):
            s = self.queue[client]["normal"].pop()
            js = s["script"]
            self.queue[client]["count"] -= 1
        elif (len(self.queue[client]["low"]) > 0):
            s = self.queue[client]["low"].pop()
            js = s["script"]
            self.queue[client]["count"] -= 1
        return dict(script=js, more=self.queue[client]["count"])

class NodeLink:
        def __init__(self, JScontroller, nodeID, slot="root", 
priority="normal"):
                self.__dict__['_JScontroller'] = JScontroller
                self.__dict__['_nodeID'] = nodeID
                self.__dict__['_slot'] = slot
                self.__dict__['_priority'] = priority
                self.__dict__['style'] = dict()

        def __setattr__(self, name, value):
            self.__dict__[name] = value
            if not name.startswith("_"):
                jval = simplejson.dumps(value)
                if (name == "innerHTML"):
                    script = "$('"+self._nodeID+"').innerHTML="+jval
                else:
                    script = "setNodeAttribute('"+self._nodeID+"', '"+name+"', 
"+jval+")"
                self._JScontroller.push(script, slot=self._slot, 
priority=self._priority)
```

The basic CherryPy Controller setup to start implementing this:

Controller.py

```
from cherrypy import session
from turbogears import controllers, expose, redirect
from time import sleep
from JSObject import *
js = JSController(slots=dict(root=[], status=[], info=[], playlist=[], 
browse=[]))
class Root(controllers.RootController):

    def myID (self):
        me = session.get("clientID", None)
        if (me == None): raise redirect("index")
        else: return me
    @expose(template="mpdwebamp2.templates.index")
    def index(self):
        me = session.get("clientID", None)
        if (me == None):
            me = js.addClient()
        else:
            js.clear(me)
        session["clientID"] = me
        return dict(script=js.baseScript)

    @expose(allow_json=True)
    def updates(self):
        me = self.myID()
        i = 0
        s = js.nextInQueue(me)
        while (len(s['script'])==0 and i < 200):
            sleep(0.01)
            s = js.nextInQueue(me)
            i += 1
        return s

    @expose()
    def connect(self, slot):
        me = session.get("clientID", None)
        if (me == None):
            me = js.addClient()
            cherrypy.session["clientID"] = me
        js.connect(me, slots=[slot])
        return "OK"

    @expose()
    def disconnect(self, slot):
        me = session.get("clientID", None)
        if (me != None):
            js.disconnect(me, slot=slot)
        return "OK"
```

This is all in the very early stages of planning right now. What I have so far is a working prototype that will most likely grow into much more. The asynchronous communications is just the first item that I will be reworking for MPD WebAMP 2. I have found that what may be appropriate for a static web page with some AJAX in it is not as appropriate when you apply the same techniques to a single page application that happens to render in a browser.

I also intend to remake the project in a more modular fashion, so that it will be a web based music player framework, with MPD as just the first backend written. Since all of the MPD interactions are going to be wrapped within a class and not accessed directly anymore, someone could easily drop in a module exposing the same API for another backend, such as XMMS2, or even a direct connection to something like GStreamer or Phonon with a custom database. With the first version, I was mostly concerned with getting a working application as quickly as possible. This time I already have a good application, so I can take my time creating a better one. I have many ideas to not only minimize the amount of data passed back and forth, but to make the framework easy for developers to work with and expand on. My goal with version 2 is not to be the best *web interface* to a music player, but the best music player.

If you would like to help with the development of this project or with JSObject in any way, please visit http://groups.google.com/group/webamp-2. You can help out by contributing code, documentation, artwork (backgrounds, buttons, toolbars...), testing, or just let me know your opinions and suggestions.

EuroPython 2007

Paul Boddie, David Boddie and other EuroPython2007 Wiki Contributors

Now in its sixth year, EuroPython - known more formally as the European Python and Zope conference - moves on once again. Upholding the EuroPython tradition of helping European Python users and others to learn more about Python and its uses, the conference is also a great opportunity to become acquainted with different parts of the European continent. This year, EuroPython descends upon Vilnius, the capital city of Lithuania, calling it home from Monday 9th July until Wednesday 11th July 2007. Although the programme for this year's conference has yet to be finalised, reports and materials from last year's highly successful event may be persuasive for those considering making EuroPython 2007 part of their schedule.

CERN – The European laboratory for particle physics research
http://flickr.com/photos/mrtopf/182294647/in/set-72157594184370918/

EuroPython 2006 was held at one of Europe's most significant scientific sites: CERN (the European laboratory for particle physics research) in Geneva, Switzerland. With superb facilities and a rich programme of highly informative, entertaining and inspiring talks, along with an unparalleled backdrop of seriously big (and historic) scientific hardware, the conference proved to be at least as memorable as those in previous years. As anticipated, Python's benevolent dictator for life, Guido van Rossum, provided one of the conference keynotes, choosing to focus on Python 3000 and the road ahead for the language. The other keynote provided fresh insights from a different perspective on computing, with the legendary figure of Alan Kay (personal computing and Smalltalk pioneer), unable to attend in person, appearing via a video link on the big screen in CERN's main auditorium. Presenting a selection of demonstrations and insights into the role of computing technology in the education of younger children, he highlighted the One Laptop Per Child project and its high Python content as an opportunity to introduce children to a more interactive and inquisitive form of learning than is often employed in schools around the world.

"In the middle of the Conference Room. Behind Guido van Rossum you can see the layout of the CERN Conference Building with the 4 conference rooms" http://www.flickr.com/photos/horstjens/183262607/

Many informative talks and presentations provided the focus for the conference, demonstrating Python's suitability in a wide range of environments from Web programming to graphical user interfaces, collaborative development practices and personal information management applications. These even included a talk about Indico, the basis for EuroPython's conference management site[26]. More advanced technical themes, such as version control, scientific computing and visualisation, were also covered. Talks about Python and its implementations were not excluded, and the European Union-funded PyPy project maintained a presence after the official end of proceedings to run one of many "sprints". These focused development efforts encourage collaboration between

Reinout van Rees http://flickr.com/photos/mrtopf/22795535/in/set-502414/

26 http://indico.cern.ch/conferenceDisplay.py?confId=44

experts and beginners with the aim of making improvements to projects, raising levels of expertise and providing hands-on experience to new recruits.

Of course, EuroPython is not just about talks and sprints. As is often noted by attendees at conferences like Euro-Python, the social aspects of such a gathering can often provide unforeseen value in the conversations and discussions which take place between talks or at the planned social events. Spontaneous and seemingly incon-sequential chat can often reveal details of fellow participants' expertise or experiences that would not be exposed through the talk programme alone. Although it is possible to make important contacts and exchange expertise without attending any of the talks, we wouldn't in any way recommend such an extreme approach!

Registration for EuroPython 2007 should already be open by the time you read this, and for those of you not already convinced of a journey to Lithuania this July, a talk schedule should be available in the near future. With luck, you will be able to peruse the final list of talks and still be able to register early for the conference, earning yourself a discount on the registration fee. Find out more and start planning your trip on the EuroPython Web site: http://www.europython.org/

ctypes. ctypes run!

Alex Holkner

This paper was originally presented at the Open Source Developer's Conference, which ran 5-8 December, 2006 in Melbourne, Australia. It was reviewed at that time, and selected for publication here as being of an excellent standard.

One of the new features of Python 2.5 is the introduction of ctypes as a standard library module. At the simplest level, ctypes adds the standard C types to Python: signed and unsigned bytes, shorts, ints and longs; as well as structs, unions, pointers and functions. At run-time it can load a shared library (DLL) and import its symbols, allowing a Python application to make function calls into the library without any special preparation. ctypes can be used to wrap native libraries in place of interface generators such as SWIG, to manipulate memory and Python objects at the lowest level, and to prototype application development in other languages.

This paper begins with a quick introduction to ctypes, shows some advanced techniques, and describes some examples of how it has been used by the author in his recent work.

Introduction to ctypes

Using C types in Python

ctypes is standard with Python 2.5, and can be installed as an extension module for Python 2.4 and earlier (see the references at the end of this paper for the download site).

Importing ctypes gives you access to the standard C types that are not available in Python.

```
>>> from ctypes import *
>>> i = c_ubyte(255)                       Create an unsigned byte with initial value 255.
>>> i
c_ubyte(255)
>>> i.value                                The "value" attribute gives an int, and can be modified.
255
>>> i.value += 1
>>> i.value
0                                          Attempting to set the byte to 256 wraps the value to 0
```

Notice that the value of a c_ubyte is limited to the range [0, 255], and the overflow behaviour is identical to that in C. The ranges of these types depend on your platform, but ctypes also provides known-width datatypes c_int8, c_int16, and so on. The simple numeric types are shown below:

C type	*Unsigned*	*Signed*
char	c_ubyte	c_char, c_byte
short	c_ushort	c_short
int	c_uint	c_int
long	c_ulong	c_long
long long	c_ulonglong	c_longlong
float		c_float
double		c_double

You can use c_void_p as a void pointer type, and c_char_p as a char pointer type. ctypes interprets a c_char as a byte representing a Python string of length 1, and c_char_p as an NULL-terminated string.

Arrays can be constructed by first defining the array type and then instantiating:

```
>>> my_array_type = c_int * 5               Define an int array type of length 5.
>>> array = my_array_type(1, 2, 3, 4, 5)    Create an actual array with some initial data.
>>> array
<__main__.c_int_Array_5 object at 0x2aaaaab582d0>
>>> list(array)
[1, 2, 3, 4, 5]                             The contents of the array can be seen by making a list.
```

A pointer type can be created for any basic type:

```
>>> i = c_int(42)                           Create an int with initial value 42.
>>> my_pointer_type = POINTER(c_int)        Create a type int*.
>>> ptr = my_pointer_type(i)                Create an int* pointer to i.
>>> ptr
<ctypes.LP_c_int object at 0x2aaaaab58450>
>>> ptr.contents
c_int(42)                                   The "contents" attribute of the pointer resolves indirection.
```

As a shortcut, you can create a pointer to an instance without creating the type first:

```
>>> i = c_int(42)
>>> ptr = pointer(i)
```

Unlike, C, ctypes is strict about passing arrays of different lengths and pointers to functions that expect otherwise. For example, ordinarily you cannot pass an array to a function expecting a pointer. You can make explicit casts between pointers and arrays when you need to:

```
>>> a = (c_int * 5)()                       Create an array of int of size 5.
>>> list(a)
[0, 0, 0, 0, 0]                             The array is initialized to zeros by default.
>>> ptr = cast(a, POINTER(c_int))           Cast the array to a pointer to int.
>>> ptr.contents
c_int(0)                                    The contents of the pointer is the first element of the array.
>>> ptr.contents.value = 13
>>> list(a)
[13, 0, 0, 0, 0]                            Changing pointer's contents affects the original array.
```

Defining a struct type is done by subclassing Structure:

```
>>> class Point(Structure):                 This struct is equivalent in C to:
...     _fields_ = [                            typedef struct {
...       ('x', c_short),                               short x;
...       ('y', c_short)                                short y;
...     ]                                       } Point;
...
>>> sizeof(Point)
4
>>> p = Point()
>>> p.x
0
```

The struct will follow the de-facto rules for packing as in C, but member alignment can be overridden if necessary. Union types can be defined in a similar way, and structs and unions can be nested, even anonymously.

Calling library functions from Python

ctypes can read symbols and function entry points from Windows DLLs, Unix shared libraries and OS X Mach-O objects and frameworks. You can directly specify the name of a library to load, but it is more platform neutral to let ctypes locate it for you:

```
>>> from ctypes.util import find_library
>>> find_library('SDL')                     SDL is an open-source cross-platform media library.
'libSDL-1.2.so.0'                           On Linux.
'/Library/Frameworks/SDL.Framework/SDL'     On OS X.
'C:\\WINDOWS\\system32\\SDL.dll'            On Windows.
```

Once you have obtained the name of the library, you can load it. Symbols will be imported on demand. For example, on Unix systems we can load libc to get the standard C functions:

```
>>> libc = cdll.LoadLibrary(find_library('c'))       find_library returns 'libc.so.6' on Linux.
>>> libc.strcmp('alice', 'bob')
-1
```

Notice that ctypes automatically converts Python string objects to c_char_p, and assumes the return value of the function is c_int, which it converts to a Python int. You can explicitly tell ctypes the return type and argument types of a function by setting the restype and argtypes attributes:

```
>>> libc.strcmp.restype = c_int
>>> libc.strcmp.argtypes = [c_char_p, c_char_p]
```

In this case the only benefit is that ctypes will now raise an exception if you pass incorrect arguments to strcmp. For other functions, however, it is often necessary to describe the argument and return types.

On Windows, the system libraries are readily available and can be used directly:

```
>>> windll.kernel32.GetTickCount()
469494577
```

Python callback functions

ctypes allows Python functions or bound methods to be used as callbacks from library functions. To do this, create a type for the callback function specifying its arguments and return type, then create a callback by instantiating this type with your function. This example uses the standard C qsort (quicksort) function, using a Python function as the comparator:

```
>>> compare_type = CFUNCTYPE(c_int, POINTER(c_int), POINTER(c_int))
>>> def compare(ptr_a, ptr_b):
...     return cmp(ptr_a.contents.value, ptr_b.contents.value)
...
>>> libc.qsort.restype = None
>>> libc.qsort.argtypes = [c_void_p, c_size_t, c_size_t, compar_type]
>>> data = (c_int * 5)(5, 2, 3, 1, 4)
>>> list(data)
[5, 2, 3, 1, 4]
>>> libc.qsort(data, len(data), sizeof(c_int), compare_type(compare))
>>> list(data)
[1, 2, 3, 4, 5]
```

You must take care to keep a reference to the function type instance alive for as long as the library will use it—neglecting this was the most common error made by the author in recent work.

Advanced uses of ctypes

Pointer arithmetic

Pointer arithmetic can be useful, for example, for passing just part of a large array to a library function. This is not supported in ctypes, in that addition is not overloaded and the internal address for a pointer is not directly modifiable. There are several ways around this, however. One way is to use the from_address method, which is used to construct a ctype instance from a given memory address:

```
>>> def ptr_add(ptr, offset):
...     address = addressof(ptr.contents) + offset
...     return pointer(type(ptr.contents).from_address(address))
...
>>> data = (c_int * 5)(1, 2, 3, 4, 5)
>>> ptr = cast(data, POINTER(c_int))
>>> ptr.contents
c_int(1)
>>> ptr = ptr_add(ptr, 3 * sizeof(c_int))
>>> ptr.contents
c_int(4)
```

Manipulating CPython objects

In the standard implementation of Python, all objects are represented internally with a struct that extends PyObject. Manipulating custom types introduced by extension libraries is possible by creating a ctypes type to match the extension type. Such a type should begin with the equivalent of PyObject_HEAD. For example, the Numeric struct is given as:

```
class _Numeric_PyArrayObject(Structure):
    _fields_ = [('ob_refcnt', c_int),
                ('ob_type', c_void_p),
                ('data', c_void_p),
                ('nd', c_int),
                ('dimensions', POINTER(c_int)),
                ('strides', POINTER(c_int)),
                ('base', c_void_p),
                ('descr', c_void_p),
                ('flags', c_uint),
                ('weakreflist', c_void_p)]
```

You can’t instantiate this type directly from a Numeric array, but fortunately the id() function returns the address of any object (this is not documented behaviour). Now you can directly manipulate fields of the Numeric array, such as its base pointer, which is otherwise not possible.

```
>>> import Numeric
>>> data = Numeric.array([1, 2, 3, 4], Numeric.Int32)
>>> data_obj = _Numeric_PyArrayObject.from_address(id(data))
>>> libc.free(data_obj.data)
>>> libc.calloc.restype = c_void_p
>>> data_obj.data = libc.calloc(10, sizeof(c_int32))
>>> data_obj.dimensions.contents.value = 10
>>> print data
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0],'i')
```

In this example, the base pointer of a Numeric array is changed to a buffer we create ourselves. This is fairly contrived, in that it does nothing that you couldn’t do just by creating a new array, but it does show how to overlay a Numeric array over any preexisting data, for example, a buffer returned by another library—something not possible with Numeric alone.

Automatic wrapper generation

ctypes_codegen is an experimental utility available from the ctypes project page. It will read one or more header files for a library and write a Python module that includes all the necessary structs, unions, constants and function prototypes for direct use.

Examples

Prototype application development

ctypes makes it very practical to develop your C applications by prototyping. Instead of beginning with the main() function and all the required utility of parsing command-line arguments, reading data files and formatting output, start instead with the critical algorithms, and write support code in Python.

```
int complex_algorithm(int a, int b)
{
    return a * b;
}
```

```
gcc -fpic -shared -o alg.so alg.c
```

```
>>> alg = cdll.LoadLibrary('./alg.so')
>>> alg.complex_algorithm(6, 7)
42
```

Even if the final product will not use Python, the programming overhead of using ctypes is so low that the advantages of rapid development and interactive debugging will make it easily worthwhile.

Pygame and SDL

Pygame is a Python extension that provides graphics, audio and input functions for creating games and multimedia applications. Internally it uses SDL for cross-platform media support.

Earlier this year the author reimplemented Pygame as a pure Python module using ctypes as a project sponsored by Google Summer of Code. The project began by writing ctypes definitions for all of the SDL structs and functions, and the Pygame functionality was rewritten over the top of this layer. Automatic code generation was not used, as there were many cases in which a more “Pythonic” API was required than the C one.

Performance was a major concern in this project, as image processing is something that Python generally does slowly. For simple processing and conversion, regular expressions are used on the image buffer, which run surprisingly fast – comparable to native C code. For more advanced processing the Python Imaging Library (PIL) was used. In the future it is envisioned that many of these functions will have multiple implementations making use of pure Python, PIL, native code and PyRex.

The Long Term

The C implementation of Python, now at version 2.5, is becoming just one in a community of Python implementations. Jython (for Java), IronPython (for the .NET platform) and PyPy (a statically and JIT compiling monster) are all progressing quickly and are becoming ready for production use. Standard Python extensions, including those generated by SWIG, cannot be used with these implementations, but they are all expected to support ctypes in the future if they do not already. When this is the case, native code can be used by any implementation of Python with a single ctypes wrapper module.

For this reason, as well as the simplification of development and maintenance, large projects are being reimplemented using ctypes. SDL and Pygame have been discussed, and recent work on OpenGL-ctypes is nearing completion.

References

ctypes project page

http://starship.python.net/crew/theller/ctypes/

Google Summer of Code
http://code.google.com/soc/

Pygame project page
http://www.pygame.org/

Pygame-ctypes and SDL-ctypes project page
http://www.pygame.org/ctypes

OpenGL-ctypes project page
http://pyopengl.sourceforge.net/ctypes/

A Rails / Django Comparison

Ben Askins and Alan Green

This paper was originally presented at the Open Source Developer's Conference, which ran 5-8 December, 2006 in Melbourne, Australia. It was reviewed at that time, and selected for publication here as being of an excellent standard.

Abstract

Ruby on Rails (“Rails”) is the dominant web programming framework for Ruby and, even outside the Ruby community, is considered the epitome of the latest generation of high-productivity, open source web development tools. Django is one of many competing web development frameworks for Python. It is notable, first, for being highly regarded amongst Python programmers, and second, for being one of the few of the new generation of framework that does not ape Ruby on Rails. Both Rails and Django claim greatly enhanced productivity, compared with more traditional web development frameworks.

In this paper, we compare the two frameworks from the point of view of a developer attempting to choose one of the two frameworks for a new project.

1. Introduction

Ruby on Rails (“Rails”) is the dominant web programming framework for Ruby and, even outside the Ruby community, is considered the epitome of the latest generation of high-productivity, open source web development tools. Django is one of many competing web development frameworks for Python. It is notable, first, for being highly regarded amongst Python programmers, and second, for being one of the few of the new generation of framework that does not ape Ruby on Rails.

This paper is written for developers wanting to choose one of these two frameworks for their own development. For the purposes of this comparison, the authors wrote a specification for a small web application that they implemented twice: once by Ben in Rails, and once by Alan in Django. The implementations were then compared, both quantitatively by code size and implementation time, and with more qualitative measures such as examining the “flavour” of the HTML templating languages.

There are of course other areas that may also be relevant to developers choosing between Rails and Django which are not examined in this paper. This includes relative and absolute performance, amenity to deployment in an enterprise production environment, and availability of third-party applications and components.

The source code for both the Django and the Rails applications are available in a subversion repository at http://3columns.net/habitual/.

1.1 Specification Overview

The small application the authors implemented is named “Habitual Readers.” It purports to be the public website of a book reading club, allowing viewers to see which club memebers have read which books, and their comments upon each read book. The application also categorises books using tags, and retrieves information from Amazon [*http://www.amazon.com/ -Ed*]. Along with the specification, the authors also developed a static HTML prototype of each of the application's seven public pages. The basic HTML and CSS layout is a modified version of a template by Andreas Viklund [VIKLUND].

The Habitual Readers specification requires that the application allow logged readers to add, change and delete books, readers, reading occasions (i.e. An occasion when a reader read a book) and tags. It does not specify the look and feel of these “admin” pages.

1.2 Habitual Readers Implementation

We developed the software on our own PCs, on a part-time basis, recording the time spent on each development task. We each used the latest version of each web framework - "Edge Rails" for Ruby and "SVN trunk" for Django – as of August, 2006.

Approaching the implementation of this application, Ben and Alan had approximately equivalent experience in Rails and Django. Ben has been working in Ruby and Ruby on Rails on a part time basis for six months. Ben has also been developing in various database-backed software environments since the earliy nineties. Alan had only two months of part-time Django experience, though he had been developing web applications in various Python frameworks since 2003, and in other languages since 1997.

At the conclusion of the implementation, we noted the following variation from the specification:

1. The Django application includes an extra "/" at the end of each URL. This behaviour cannot be easily changed without affecting the Django admin application.

2 Quantitative Comparison

We compared the two application implementations using two convenient measures: lines of code and implementation time. These quantitative measures were then used to inform the analysis that accompanies the qualitative comparisons below.

2.1 Lines of Code

In producing these counts, we included all files containing a large proportion of hand-written code. Rails and Django generate configuration files in their native language – Ruby or Python – which the developer is required to modify. We did not include these configuration files. Python and Ruby lines of code were measured using David A Wheeler's "sloccount" tool [WHEELER], which ignores blank lines and comments. HTML line counts were produced by running the "wc" shell command on each implementation's template files.

	Rails	***Django***
Ruby / Python Model Code	83	116
View / Controller Code	203	109
HTML Helpers/ Template Tags	56	26
Schema Migration	118	-
YAML Data Loading	26	69
Authentication	31	-
Ruby / Python subtotal	517	320
Templates	297	406
Totals	814	726

Comparing primary implementation languages, the Django application contains one third fewer lines of code than the Rails application. The difference would have been larger if we had not implemented YAML data loading in the Django implementation, as the YAML data loading counts for 20% of the lines of code in the Django application. (YAML is "Yet Another Markup Language", which has a syntax convenient for specifying test data [YAML]).

There is also a large difference in the number of lines of template code. The Django templates have one third again as many lines as the Rails templates, even though the Rails templates encompass the admin pages while the Django tamplates do not.

2.2 Implementation Time

The authors recorded the time they took to implement the Habitual Readers application, with the results presented in the table below. We have recorded time in three columns: time spent on the Rails implementation, time spent on the Django implementation, and time spend on tasks that benefit both projects. All times are measured in elapsed hours and minutes.

Task	***Rails***	***Django***	***Common to Both***
Initial HTML Protoype			4:30
Develop Test Data			1:36
Project Set Up	0:15	0:11	
Models	1:30	0:09	
Home Page	3:00	1:40	
Basic Pages	5:00	2:08	
Admin Pages	8:25	:57	
Amazon Interface	1:00	2:18	
Data Loading (Code)		1:36	
Test, Review and Tidy	1:30	1:31	
Totals	**20:40**	**10:30**	**6:06**
Totals including "Common times"	**26:46**	**16:36**	

It was clearly faster for Alan to implement this application in Django than for Ben to implement this application in Rails. The bulk of this difference can be attributed to the extra effort required to implement the administration functions in Rails which accounts for approximately seven and a half hours. Excluding the admin pages, and allowing for factors such as the variability in work environments and the experience of each developer, the implementation times are approximately equal.

3 Qualitative Comparison

This section compares a selection of attributes of Rails and Django.

3.1 HTML Templating

HTML templates are a core feature of both Rails and Django. Both allow templates to be composed of a base template that defines the overall layout of the page, plus individual page templates that define the specific content of the page. In Rails, the base template is called a "layout" while the individual page templates are "views". In Django, both are plain templates.

The key difference between the framework is the way they embed dynamic content. Rails views use in-line Ruby code fragments, and may therefore contain arbitrarily complex functionality. Rails encourages developers to be as pithy in their view code as they are in their controller or model code, rewarding aggressive refactoring with a satisfyingly concise result. Django, in contrast, uses a simpler templating language conceived specifically so that web designers with HTML skills, but minimal programming knowledge, can build templates.

For example, on the main page of Habitual Readers, there is a tag cloud – a list of links to tags, sized depending on the popularity of the tag. The Rails application retrieves the list of tags from the database and makes it available to the view. The section of the view that renders these tags is:

```
<%= render :partial -> 'tag cloud' %>
```

This refers to a Ruby partial, a fragment of a page, which is contained in `_tag_cloud.html`. That file's content is:

```
<h3>categories</h3>
<p class='taglist'>
      <%= render :partial -> 'tag', :collection => @tags %>
</p>
```

This generates the heading, then refers to another partial, which will be used once for each object in the collection named `tags`. The `_tag.rhtml` file contains a single line:

```
<%= link tag[:name], tag_url(tag[:name]), :class => “level_#{tag[:pop_level]}” %>
```

This call a Ruby helper function to generate a link to the tag, setting the CSS class appropriately.

By contrast, the portion of the Django template that generates the output is:

```
<h3>categories</h3>
<p class=”taglist”>
{% for tag in tags %}
      <a class=”level+{{ tag.pop_level }}”
            href=”{{tag.get_absolute_url }}”>{{ tag.name }}</a>
{% endfor %}
</p>
```

The Rails views appear more complex than the Django templates, spreading the HTML template to generate the tag cloud across three separate files. The advantage of the Rails approach is that it allows the developer to reuse even the smallest fragment of code, resulting in significantly less repetition and fewer overall lines of template code, even on a small application such as Habitual Readers. Rails doesn't force this approach, however, allowing the developer to produce a near identical view to the Django template as demonstrated in the following example:

```
<h3>categories</h3>
<p class=”taglist”>
<% @tags.each do |tag| %>
      <%= link_to tag[:name], tag_url(tag[:name]), :class => “level_%#{tag:\
            popup:level]}”
%>
<% end %>
</p>
```

The more explicit approach taken by Django is considered simpler to teach to web designers without any background in software development [CROFT].

3.2 Schema Generation and Evolution

Model objects are a central concept in both Rails and Django, through each take a different approach in their implementation. Rails also provides facilities for schema evolution, which Django does not.

3.2.1 Defining Model Classes

Rails implements the active record patterns [FOWLER, M] as its data access layer. To define a model, the developer derives a new class from the ActiveRecord base class. ActiveRecord deduces the model's attributes from the table in the database that matches the plural of the class name. The developer can modify this mapping, but it is generally considered good Rails practise to keep such modifications to a minimum.

Django, on the other hand, requires the developer to explicitly specify not only each class, but also each attribute. Django has the standard tools for creating a database schema from an application's model definition.

A model class from the Habitual Readers application serves as an example. Instances of the ReadingOccasion class record an occasion when a reader read a book. Django code explicitly defines each attribute of the model, including metadata necessary to define the underlying database table.

```
class ReadingOccasion(models.Model):
      reader = models.ForeignKey(Reader)
      book = models.ForeignKey(Book, edit_inline=models.STACKED,
            num_in_admin=1)
      finished = models.DateField(core=True)
      reading_time = models.FloatField(core=True, max_digits=5,
            decimal_places=2, blank=True)
      notes = models.TextField(maxlength = 2000, blank = True)
```

The Django model also includes a small amount of metadata used to specify its appearance in the automatically generated admin pages, which are explained below.

By contrast, the Rails version is minimal. It defines only the relationships between itself and other models. All of the attributes of the model (when the book was read, how long it took to read and additional notes) are added when Rails examines the table definition at runtime:

```
class Reading < ActiveRecord::Base
      belongs_to :book
      belongs_to :reader
end
```

In comparing the two, we see that the Rails class is much shorter, with the trade off being that the model's attributes are not documented in the class definition. Django, on the other hand, requires that the developer define the entire model, including its attributes, within the class definition. This is another example of Rails choosing the concise over the explicit while Django has chosen the converse.

3.2.2 Evolving Model Classes

While Django has facilities for creating a database schema from the model definition, it does not provide the developer with any support for modifying (or, “evolving”) the model definitions while preserving data in the underlying database tables. The Rails migrations mechanism addresses creating and evolving model classes and the underlying scheme while preserving data.

A migration is a Ruby script that defines additions, modifications and deletions to the database schema. Optionally, the developer can also specify how data is to be migrated from the old version of the schema to the new version. Each migration script is assigned a version number, and changes to the schema can be rolled backwards and forwards by applying and reversing the migration versions in sequence.

Here is the migration used in the Rails Habitual Readers application to create Readings table:

```
class CreateReadings < ActiveRecord::Migration
      def self.up
            create_table :readings do |t|
            t.column “book_id”, :integer
            t.column “reader_id”, :integer
            t.column “date_read”, :datetime
            t.column “reading_time”, :integer
            t.column “notes”, :text
      end

      def self.down
            drop_table :readings
      end
end
```

The `up` method is invoked to apply the migration. It will create a table named `readings`, with the given columns. The `down` method is invoked to reverse the migration – in this case dropping the `readings` table. In understanding migrations, it is important to note that the only effect of the migration script is the modification of the schema when the script is run. When the Rails application is executing, the model class reads the database schema to dynamically determine its attributes at runtime.

There are two key advantages to Rails' incremental migration compared with Django. First, Rails provides a standard mechanism mfor deploying new releases to already running production systems while preserving data. For example, if a databas column's type is changed from char to integer, the accompanying Rails migration script would specify the steps required to move the data from the old char column to the new integer column. To perform similar operations in Django, the developer would need to write an ad-hoc SQL script.

The second advantage is that being easily rolled back, migration encourages a certain amount of experimentation with the model classes and database schema. Certainly some experimentation with models is possible in Django, especially if the model code is kept under source code control. However, as data is not preserved through such changes, it is less attractive unless there is a mechanism for quickly loading test data.

At the time of writing, the Django development community is working toward introducing a schema evolution mechanism.

3.3 Automatically Generated Admin Pages

Many web applications have a group of “admin” pages, pages used by a small number of trusted users, perhaps to enter content for publishing to a wider audience, or maintaining reference tables.

A clear area of difference between Rails and Django is Django's automatincally generated admin pages, which can save a significant proportion of development time. For example, in the Habitual Readers implementations, the development of admin pages in Rails took 29% of the development time, compared to 6% for Django.

When we write that the Django admin pages are “automatically generated”, we mean that Django generates them with only small hints from the developer. The developer gives these hints by adding attributes and parameters to the Django model classes and their fields. Despite being a little fiddly, this process is rather quick and the generated pages are suitable in a wide range situations. To further customise the look and feel, the developer may provide Django with alternate templates.

Rails chooses not to implement an administrative interface based on a distinction drawn between application infrastructure and application functionality. There are a number of Rails plugins that aim to fill the same need as the Django admin application. Two such plugins are AutoAdmin [AUTOADMIN] and Streamlined [STREAMLINED] (both in the early stages of development). Developers using Rails in their own projects are advised to check on the status of these plug-ins before commencing projects that require an administrative interface.

3.4 Internationalisation

Although we did not address internationalisation with the Habitual Readers application, this is a well-documented and much-discussed issue in both the Rails and Django communities.

In its current release, some Ruby core libraries, for example String, aren't unicode aware. This means that string operations such as length and split won't function as expected when working with non-Western text. The Rails core team have addressed this problem by merging the multibyte_for_rails plugin with the Rails ActiveSupport module as ActiveSupport::Multibyte [RAILSMULTIBYTE]. The upshot of this is that as of version 1.2, the Rails framework will extend the problematic Ruby core libraries to make them unicode-aware.

Python has had Unicode support and GNU gettext [GETTEXT] style internationalisation since version 1.6, which was released in 2000. The Django framework builds on this with standardised mechanisms for internationalising Django applications and localising responses for individual HTTP requests.

3.5 Integrating Third Party Code

Rails and Django can each take advantage of two flavours of extensions. The first is what Rails calls “plugins” and Django calls “applications”. This kind of extension is aware that it is running inside of its respective framework. The second is third party libraries written in Ruby or Python, but that aren't specifically developed for use within either framework.

Rails plugins are discrete collections of code that can be included within a Rails application. The Habitual Readers Rails implementation took advantage of the acts_as_taggable plugin to provide support for tagging books, and the acts_as_attachment plugin to support uploading a reader image and resizing it to a manageable thumbnail for use in the application. At the time of writing, 419 Rails plugins are available for download from http://www.agilewebdevelopment.com/.

Django applications can be used in a manner similar to Rails plugins. A Django site may be composed of several applications, each of which may contain model code, templates, template tags and plain Python libraries. Typically one application will contain the site's main functionality with additional applications providing features such as comments and admin pages. At the time of writing, there is no central repository of re-usable, third-party applications.

Rails and Django freely integrate with a wide range of native libraries – that is, libraries written in Ruby and Python respectively. For example, in the Habitual Readers application, both the Rails and Django implementations communicate with Amazon via native third-party libraries.

3.6 AJAX

While neither the Rails nor the Django implementations of the Habitual Readers application took advantage of AJAX, it has become so prevalent in web applications that it's worth mentioning here.

Rails includes a number of helper functions that assist with sending and responding to XmlHttpRequests. Using RJX, a Rails developer can write a template that responds to a request by generating Javascript that will be sent back in the response object and executed in the browser. Prototype and Scriptaculous helper methods are available that allow functions from those Javascript libraries to be used in RJS templates. The packaging of these Javascript libraries with Rails does not preclude the developer from choosing to work with Javascript libraries.

In contrast, Django includes only a JSON module, leaving Javascript code and the choice of a Javascript library, if any, to the developer. The Django core team argue that this is a strength of Django. Other Django developers, however, have indicated that they would prefer the Rails approach of an officially sanctioned Javascript library and associated helper functions.

3.7 Other Considerations

In addition, the following non-technical areas may be of concern to developers choosing between the two frameworks.

3.7.1 Maturity

Both frameworks were extracted from web applications that were developed in the 2003-2004 period. Rails was released to the public in July 2004, and Django in July 2005. As such, Rails has had a head start in getting community contributions to the framework and reached the milestone 1.0 release in December 2005. The current release of Django is 0.95, and there may be further changes to the Django API before the 1.0 milestone release is reached [DJANGOAPI]. There are currently 12 core team members who have commit rights on the Rails repository [RAILSCORE], and 4 on the Django repository [DJANGOCORE].

3.7.2 Market Position and Hype

A search on job sites JobServe [JOBSERVE] and Seek [SEEK] shows Rails turning up in job requirements more often than Django at a ratio of 6:1, while Python shows up more often than Ruby at a ratio of approximately 4:1.

While the Python language has a higher demand than Ruby, the Rails framework is more firmly established in the marketplace than Django. Compared to Java and J2EE, Rails and Django are both young when it comes to gaining acceptance in the marketplace.

Similar comparisons can be made using tools such as Google Battle [GOOGLEBATTLE], Ice Rocket [ICEROCKET] and Technorati [TECHNORATI] where Rails consistently comes out on top in the hype stakes.

3.7.3 Tools and Utilities

Both frameworks ship with scripts that assist in the development process and automate repetitive tasks such as schema creation and executing unit tests. Rails takes advantage of Rake, the Ruby build language to automate repetitive tasks.

Capistrano is a deployment tool that has been developed for automating the deployment of Rails applications. It executes commands on remote servers via ssh to perform tasks such as checking the latest revision of code out of a repository, running migrations and loading data. It is particularly useful when deploying to multiple production servers as it can execute commands on multiple servers in parallel. At this stage there is no such tool for Django, although Capistrano has been used by others to deploy non-Rails applications, so there seems no reason that it couldn't be used to deploy a Django application.

4 Conclusion

Django and Rails aim to solve similar problems, in a similar manner, using a similar architecture. There is no clear technical benefit for an experienced Rails development team to switch to Django or for an experienced Django development team to switch to Rails. For developers not currently working with either Django or Rails, the most important consideration is the implementation language. Ruby developers would benefit from using Rails, while Python developers would benefit from using Django, allowing them to apply skills they already have.

For developers who know neither (or both) languages, the "best" framework will depend on the development environment and type of application. The following table summarises those aspects that we have investigated in this paper:

Factor	***Rails***	***Django***
Support for model and schema evolution	Integrated framework for schema evolution.	Minimal.
Internationalisation	Some support in Rails 1.2.	Some support.
Designer friendly templates?	Possible, with the use of a third-party library.	Yes.
Third party plugin support?	Mature plugin architecture, well-used by the community.	Some support via the applications mechanism.
Javascript Support	Prototype and Scriptaculous bundled with Railes. RJS framework simplifies their use.	Possible, but no direct support for any particular library.
Flavour	Concise.	Explicit.

While choosing between these two frameworks may be difficult, the good news is that either framework is a good choice for a team wishing to develop a web application.

5 References

1. [AUTOADMIN] – http://code.trebex.net/auto-admin
2. [CROFT] – http://www2.jeffcroft.com/2006/may/02/django-non-programmers/
3. [DJANGOAPI] – http://www.djangoproject.com/documentation/api_stability/
4. [DJANGOCORE] – http://www.djangoproject.com/documentation/faq#who-s-behind-this
5. [FOWLER] – Rails Recipes, Chad Fowler
6. [FOWLER, M] – Patterns of Enterprise Application Architecture, Martin Fowler
7. [GETTEXT] – http://www.gnu.org/software/gettext/
8. [GOOGLEBATTLE] – http://www.googlebattle.com/
9. [ICEROCKET] – http://www.icerocket.com/
10. [JOBSERVE] – http://www.jobserve.com/
11. [PYAMAZON] – http://www.josephson.org/projects/pyamazon/
12. [RAILSCORE] – http://rubyonrails.org/core
13. [RAILSMULTIBYTE] – A Rails Unicode primer, http://fngtps.com/projects/multibyte_for_rails/wiki/UnicodePrimer

14. [RSPEC] – http://rspec.rubyforge.org/tools/rails.html
15. [RUBYAMAZON] – http://www.caliban.org/ruby/ruby-amazon.shtml
16. [SEEK] – http://www.seek.com.au
17. [STREAMLINED] – http://streamlines.relevancellc.com/
18. [TAGGABLE] – http://wiki.rubyonrails.org/rails/pages/ActsAsTaggablePluginHowto
19. [TECHNORATI] – http://www.technorati.com/
20. [VIKLUND] – http://andreasviklund.com/templates/
21. [WHEELER] – David A. Wheeler's Sloccount tool, http://www.dwheeler.com/sloccount
22. [YAML] – Yet Another Markup Language, http://www.yaml.org/

Upcoming Events

The following events, taken from the python.org events wiki[27], are being held between May and August this year.

May 21-25, 2007 Houston, Texas, USA: Advanced PyCamp hosted by Texas Learning and Computation Center

May 21-25, 2007 Atlanta, Georgia, USA: Python Bootcamp taught by David Beazley at the Big Nerd Ranch

June 2, 2007 Kiev, Ukraine: Exception python seminar (in Russian)

June 2-3, 2007 Paris, France: Journees Python

June 11-13, 2007 Longmont, Colorado, USA: Python training class taught by Mark Lutz.

June 16-17, 2007 Leipzig, Germany: Python course for programmers (in German) taught by Python Academy.

June 18-21, 2007 Orange, California, USA: "Python First" Workshop at Chapman University taught by Atanas Radenski.

July 9-11, 2007 Vilnius, Lithuania: EuroPython 2007

July 9-11, 2007 Melksham, England: Python course taught by Graham Ellis.

July 28-29, 2007 Leipzig, Germany: Python course for non-programmers (in English) taught by Python Academy.

August 14-18, 2007 Pasadena, California, USA: SciPy 2007: Python for Scientific Computing

August 20-22, 2007 San Francisco, California, USA: (Intensive) Introduction to Python course taught by Wesley Chun

To include your event in our next issue, or to include expanded event information, please contact us directly to ensure that your event is represented as you would like. All events available from the python.org events wiki will be included with a basic reference.

27 `http://wiki.python.org/moin/PythonEvents`

The Python Papers' Review Policy

0. Preamble
The Python Papers (ISSN 1834-3147) is intended to be both a industrial journal as well as an academic journal, in the sense that the editorial board welcomes submissions from all aspects related to the Python programming language, its tools and libraries, and community, both of academic and industrial inclinations. The Python Papers aims to be a publication for the Python community at large. In order to cater for this, The Python Papers seeks to publish submissions under 2 main streams: the industrial stream (technically reviewed) and the academic stream (peer-reviewed). This policy statement seeks to clarifythe process of technical review and peer-review in The Python Papers.

1. Right of submission author(s) to choose streams
The submission author(s); that is, the author(s) of the article or code or any submissions in any other forms deemed by The Python Papers editorial board (hereafter known as 'editorial board') as being suitable; reserves the right to choose if he/she wants his/her submission to be in the industrial stream, where it will be technically reviewed, or in the academic stream, where it will be peer-reviewed. It is also the onus of the submission author(s) to nominate the stream. The editorial board defaults all submissions to be industrial (technical review) in event of non-nomination by the submission author(s) but the editorial board reserves the right to place such submissions into the academic stream if it deems fit.

2. Right of submission author(s) to nominate potential reviewers
The submission author(s) can exercise the right to nominate up to 4 potential reviewers (hereafter known as "external reviewer") for his/her submission if the submission author(s) choose to be peer-reviewed. When this right is exercised, the submission author(s) must declare any prior relationships or conflict of interests with the nominated potential reviewers. The final decision rests with the Chief Reviewer.

3. Right of submission author(s) to exclude potential reviewers
The submission author(s) can exercise the right to recommend excluding any reasonable numbers of potential reviewers for his/her submission. When this right is exercised, the submission author(s) must indicate the grounds on which such exclusion should be recommended. Decisions for the editorial board to accept or reject such exclusions will be solely based on the grounds as indicated by the submission author(s).

4. Peer-review process
Upon receiving a submission for peer-review, the Editor-in-Chief (hereafter known as "EIC") may choose to reject the submission or the EIC will nominate a Chief Reviewer (hereafter known as "CR") from the editorial board to chair the peer-review process of that submission. The EIC can nominate himself/herself as CR for the submission. The CR will send out the submission to TWO or more external reviewers to be reviewed. The CR reserves the right not to call upon the nominated potential reviewers and/or not to call upon any of the excluded potential reviewers as suggested by the submission author(s). The CR may also concurrently send the submission to one or more Associate Editor(s) (hereafter known as "AE") for review. Hence, a submission in the academic stream will be reviewed by at least three persons, the EIC as CR and two external reviewers. Typically, a submission is reviewed by three to four persons: the EIC as CR, an AE, and two external reviewers. There is no upper limit to the number of reviews in a submission. Upon receiving the review from external reviewer(s) and AE(s), the CR decides on one of the following options: accept without revision, accept with revision, reject; and notifies the submission author(s) of the decision on behalf of the EIC.

If the decision is "accept with revision", the CR will provide a deadline to the submission author(s) for revisions to be done and will automatically accept the revised submission if the CR deems that all revision(s) were done; however, the CR reserves the right to move to reject the original submission if the revision(s) were not carried out by the stipulated deadline by the CR. If the decision is "reject", the submission author(s) may choose to revise for future re-submission. Decision(s) by CR or EIC is final.

5. Technical review process

Upon receiving a submission for technical review, the Editor-in-Chief (hereafter known as "EIC") may choose to reject the submission or the EIC will nominate a Chief Reviewer (hereafter known as "CR") from the editorial board to chair the review process of that submission. The EIC can nominate himself/herself as CR for the submission. The CR may decide to accept or reject the submission after reviewing or may seek another AE's opinions before reaching a decision. The CR will notify the submission author(s) of the decision on behalf of the EIC. Decision(s) by CR or EIC is final.

6. Main difference between peer-review and technical review

The process of peer-review and technical review are similar, with the main difference being that in the peer review process, the submission is reviewed both internally by the editorial board (EIC/CR and assigned AE(s)) and externally by external reviewers (nominated by submission author(s) or nominated by EIC/CR). In a technical review process, the submission is reviewed by the editorial board and any external review maybe at the editorial board's discretion.

7. Umbrella philosophy

The Python Papers' editorial board firmly believes that all good (technically and/or scholarly/academic) submissions should be published and that the editorial board is integral in refining all submissions. The board believes in giving good advice to all submission author(s) regardless of the final decision to accept or reject and hopes that advice to rejected submissions will assist in their revisions.