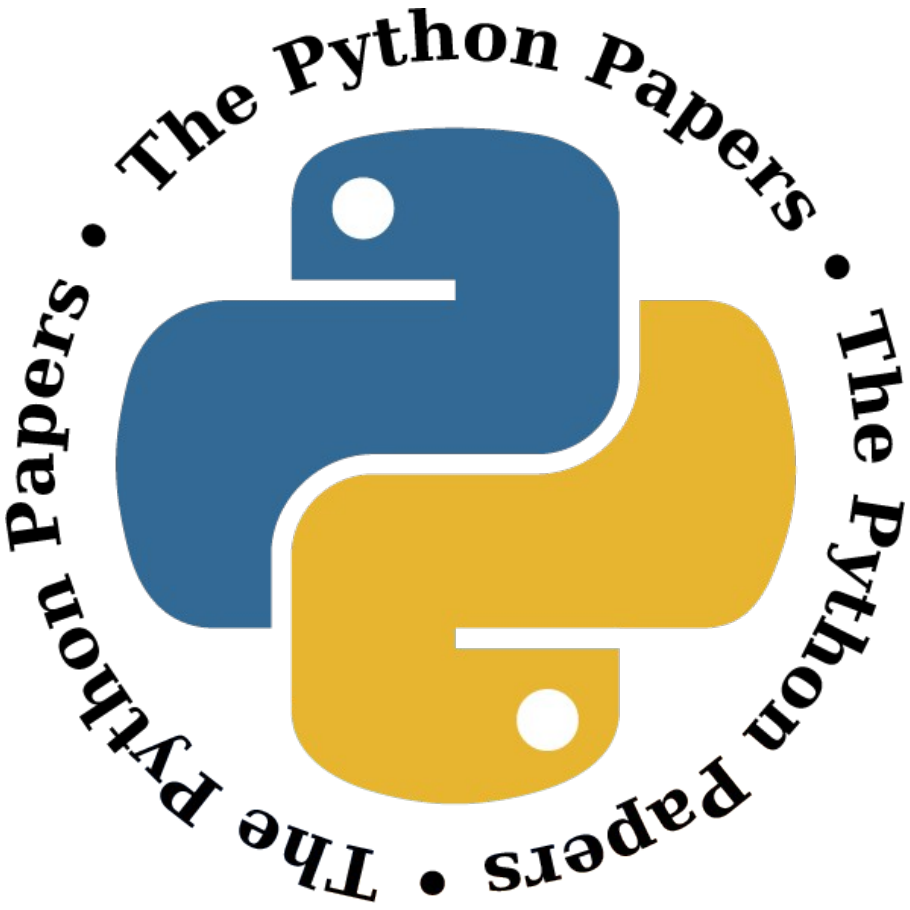
The Python Papers



Volume 2, Issue 3
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

The Python Papers was first published in 2006 in Melbourne, Australia.

Referees

An academic peer-review was performed on all academic articles. A list of reviewers will be published in each December issue. This has been done in order to ensure the anonymity of reviewers for each paper.

The Python Papers

Volume Two, Issue Three : August 2007

In this Edition:

Articles and Editorials:

Peer Reviewed Submissions:

Letter from the Editor

Tennessee Leeuwenburg

Hello to the readers of *The Python Papers*! This issue marks the end of our first full year of publications – the next edition will be our first anniversary. Our domain name is up for its yearly renewal and it seems appropriate to look at some of the things which we have achieved so far:

> 13 articles published and five academic papers;

> 18522 total site visits;

> Highlights from Python User Groups in four countries

> Connections with many individuals and groups

> The creation of our website and editorial policies

It has been a time not without its trials. Our first issue was little more than a few contributions from the editorial board and a dash of planning with regards to layout and design. From these very humble beginnings, the Python community has responded with the great contributions which make our continued existence possible. It is only appropriate that this issue look in particular detail at some of those individuals. Our feature article explores the barriers that women face in tech communities. This article applies generally, but holds a lot of valuable information for those individuals seeking to promote Python through advocacy, development or community involvement. We include our regular section on Python User Group highlights, this time examining Brisbane-PUG. Our academic section presents PyPhant, a framework for the creation and application of information flow models.

I'd also like to say a big hello to everyone at the Front Range Pythoneers in Boulder, Colorado who were kind enough to include me in one of their meetups while I was on overseas visit!

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.

Python User Group Highlights: Introducing BrisPy

David P Novakovic

BrisPy: The Python User Group Meetings

The Brisbane Python User Group is heading into its fourth month as a forum for local Pythonistas. The user group was started by Stephen Thorne from NetBox Blue in Milton and is kindly hosted by NetBox Blue themselves who provide pizza and beer for the gathering.

Brisbane at night by Chris Wallace

Milton is very close to the central business district of Brisbane. Brisbane is the capital of Queensland, Australia and is about two hours north of the most easterly point on the Australian east coast. Queensland is generally known as the warmer, sunnier state in Australia. Brisbane has a mixture of both industrial and business oriented industries, interspersed with a large tourism industry, particularly in it's neighbouring city Gold Coast. There are three major universities with many campuses. Unfortunately, Java is still the tertiary institution language of choice, so most people at the meetings are seasoned programmers who either use Python at work or wish they could. The IT industry in this area is not known for being particularly large or interesting, so many people know each other. This made the first BrisPy meeting an inevitable success.

The first meeting went will with about 15 people coming to see Stephen's talk about using scapy to monitor (sniff) network traffic. This also gave a few people from #python on irc.freenode.org a chance to meet face to face and have some good discussion. To see the slides for the presentation, head to: http://oss.netboxblue.com/pug/scapy.html

The second meeting was presented to by Clinton Roy who shared his knowledge about using ANTLR for Python. Initial comments about Java technology at a PUG were quickly dispelled as Clinton showed everyone some simple examples for generating parsers (and lexers etc) for simple grammars. All in all a good night that was very interesting, especially provoking when considering small DSL (domain specific languages). To see the slides for the ANTLR talk, head to: http://azure.humbug.org.au/~croy/antlr.pdf

Brisbane at night by Chris Wallace

At the moment the group is really keen to find any speakers to come and deliver material about a Python topic that they may know a lot about or hold dear. Some hot topics include web frameworks, ctypes, parallel python and twisted.

After each meeting discussions, pizza and beer take up the rest of the night. Often it is quite late when the meeting finishes. While discussion is normally around topics in Python, discussion also often revolves around what people are using Python for, some of these topics include:

Spam filtering

Natural Language Processing

Parsers

Web Frameworks

Testing Frameworks

Databases

For those interested in coming along to the BrisPy PUG, head to the Google Groups page at http://groups.google.com/group/brispy and register your interest. Alternatively head to the wiki for more information: http://wiki.python.org/moin/BrisbanePUG

The Barriers Women Face in Tech Communities

Gloria W. Jacobs

[*Editor's Note – This article was written for* The Python Papers*, but also co-published on the DevChix website prior to the release of this issue of* The Python Papers*. As a result of publication on the DevChix site, many comments were left discussing the contents of this article. They were in the main supportive, although many contrasting and at times opposing points of view were put forward.*

Readers who would like to comment on this article are welcome to send feedback to editor@pythonpapers.org for consideration in our next issue. As with all submissions, these will undergo an editorial process to ensure they meet with the standards of our journal.

DevChix is exactly what the article describes as a women-friendly group: a tightly moderated community for tech women to share ideas, paid gigs, and information, as well as a safe place to learn from one another and freely ask questions without harsh criticism and elitist or exclusionary comments.]

Introduction

This subject has been on the minds of many tech women for years. The issue is discussed regularly, almost cyclically at times, as we spin our collective wheels to try to find causes and solutions. I was reluctant to write about it, since I find the subject matter daunting and the problem almost insurmountable at times. But three different sources approached me simultaneously, asking for this article. This article feels as if it is manifesting through me rather than from me, as a collective opinion and observation from the many tech women with whom I've worked and spoken. So many factors are in play when discussing this issue that I can only hope to address many of them without writing a tome.

My tendencies are to pick up on patterns – in human interaction, in data, in almost everything. I am a computer science/math major, and my brain loves to seek out the non-obvious patterns in whatever I am observing. One of my favorite pastimes is to figure out broken elevator algorithms: what event causes the doors to close too quickly, how the cars are distributed amongst the people requesting the elevators, etc. One of the not-so-favorite puzzles my brain likes to do is to pick up on patterns of human behavior from both men and women which affect how tech women are treated, both on and off the job. This article is all about the patterns I and other women have found in human interaction, office and online environments, which make them less conducive to tech women participation.

The less obvious

I won't be addressing the more obvious problems affecting women in tech environments such as the pay scale gap between women and men, the blatantly inappropriate sexism and personal harassment that has taken place, and persists. My reasons are because I feel these issues have been properly and effectively addressed by other women in tech (they're not resolved by any means, but at least public awareness is rising). With this article, I am attempting to address the less obvious or un-obvious reasons why some tech environments are intolerable for many women.

The material for this article came about through my participation in both women-only and mixed gender groups of many kinds. When I wonder why tech groups aren't tolerable for many women, I look at the inverse of the problem: What makes women-only tech groups more tolerable for women? My observations follow.

Why do women-only tech groups exist?

Over the years I had participated in many different types of women-only groups. Women-only

drumming groups, women-only political groups, women-only tech groups, have all provided what women consider to be a 'safe haven' to freely learn these arts, share ideas, expose each other to paid 'gigs', and help each other accomplish tasks. Women in these groups usually had nothing else in common except for the fact that they (1) were female, and (2) shared an interest and experience in drumming/politics/tech. Their professions, ages, skill levels, hobbies, sexual orientations, life experiences, marital status, children/grandchildren/no children, everything else about these women varied vastly.

My brain began to try and pick up on patterns which would explain why all of these different types of women feel as if they need a women-only group, and what such a group can provide that a mixed gender group cannot. Here are my observations.

Community plays an important and prevalent role in women-only/women-friendly groups.

No matter the group or the reason for gathering, **all** of the women-only, and most of the successful women-friendly groups to which I have belonged had a strong sense of community. They make a tremendous effort to communicate well, to be fair with each other, and to provide support related to the groups goals, sometimes even extending outside of the groups goals.

This mindset is so common that women come to expect it when joining these groups and foster it once they have joined. The implied message is that a strong, focused, collective effort will be spent to run things fairly and treat all members equally, and collective discussion happens when this is not accomplished. This is the lure to women-only groups.

Communication style is directly affected by this sense of community

I have never seen a woman harshly criticize another woman in these groups. Never have I seen or heard anything like "You suck", "You're wrong, idiot" when women in these groups communicate. Differences are usually discussed in a civilized manner. There is the occasional strong disagreement or ousting of a member now and again, but it happens after a discussion involving the entire group, and an effort to work out their differences. I am sure harsh criticism happens somewhere in some women's groups. But I am also sure that it's not tolerated for very long by other female members.

This style of communication is directly at odds with much of the harsh criticism and disdain found in predominantly male public comments, especially in most public online tech comment spaces, unfortunately.

Destructive criticism is the best way to keep a site predominantly male. It implies that there is no concern about whether a person can learn from a response or not, or whether they would find offense. It is an outward display of ego, a territorial "pissing rite" in which most women do not and will not participate.

That being said, there are many men who flock to women-only groups for the same reasons as women. They do not want to be subjected to the predominantly male style of communication where there is no sense of community, or even just simple accountability. They grow tired of the "pissing rite", the absurd declarations of false boundaries, the outward display of insecurity through harsh criticism, implicit claims of "my way, my expertise, my right, never yours", and poor display of ego. This mode of communication is an unproductive waste of time, and many men realize this as well. "I feel at home here because I really don't want to deal with that male ego bullshit", one male member of our political group stated to me.

Men who seek out women's groups are usually welcome, or a splinter group is formed to accommodate these men, once it is determined that they do not seek membership for the wrong reasons. Some of the wrong reasons are:

1. "I will be the only male member, and will therefore have my choice of 'chicks'". Nope. It's not happening.

2. “I will be the only male member, and I’ll guide/help/protect these lost / vulnerable / endangered women”. This is not only unnecessary, but laughable. Women find the implications of these assumptions both offensive and so primitive that it is hysterically funny.

3. “I will infiltrate because I hate women, and want to try to dissolve the group in some way”. This is very rare, but happens. The good news is that the motives of both men and women who attempt this become very obvious very quickly.

Women-only/women-friendly tech groups and gatherings offer a level of awareness of and accountability for behavior not found in most mixed gender tech groups/gatherings.

Awareness of and accountability for behavior in women’s groups means a lot more than just safety from sexual harassment, or discrimination. It means that if one is treated unfairly or harshly in any manner that a person finds offensive, the entire community will hear your claim. They will give you advice, opinions, and will collectively decide if action should be taken.

There has recently been a call for all public message board admins to get tougher about removing blatantly discriminatory, harassing, or sexually objectifying comments. This is a very necessary, damned good start. But to genuinely make an online tech community women-friendly, it needs even tighter moderation against **harsh/demeaning criticism, elitist commentary, and exclusionist statements**, the three most prevalent and women-unfriendly types of communication found in almost all moderated online tech message boards. There is no better way to give women a message that their comments are not welcome than implying that: (1) this is forbidden territory, women have no expertise here; (2) your comments are stupid, wrong, or ridiculous; or (3) we’re so much smarter than you. Discussion, constructive criticism, even heated debate happens in women-only groups, but these methods of communication are avoided.

Both online and off, I have seen men who communicate this way with everyone and men who only choose to communicate this way with women. I have also seen this behavior tolerated or ignored for the most part. Here are my observations on why this happens.

Men are generally very good at ignoring bad behavior.

This is both a blessing and a curse. In my most recent office environment, we had situations where a male colleague’s behavior was abrasive in one of these three ways mentioned. “That sucks, doesn’t it?” I asked another male colleague. “Yeah, but I just ignore it. That’s just the way he is. He is always like that” He responded. This is what I’ve seen as the general male way of coping with this poor communication style.

It’s a blessing that many men can ignore it, in the sense that most men do not get caught up in deep analysis of why this person said a specific thing and what this person could have really meant, etc. When almost everything is taken at face value and not over-analyzed, the ability to ignore communication issues makes it is easier to resolve the simple issues and move on. I have seen some women in office environments do the over-analysis and take offense when there never was one given. I don’t see men do this very often and it makes communication quicker and easier.

Ignoring communication issues is also a curse because one obnoxious person is allowed complete freedom to make excessive noise, be rude and disruptive, or explicitly offensive. Most men, online or in the office, will ignore it. Most women will notice it but not say or do anything about it, for a variety of reasons which are tangential to this article. The offender often thrives on the fact that no one told them to stop, so they continue. Sometimes the offender is not socially adept enough to pick up on the fact that ignoring someone implies intolerance at some level. They somehow missed the message most three year olds learn: “I’m ignoring you because I don’t like your behavior”, so they continue the intolerable behavior.

This is so prevalent in online tech communities that it is the primary reason why many women do not participate. The poor communication and behavior of even one boorish, ego-driven, elitist, socially inept geek is simply intolerable for most women. Women generally tend to assume that everyone will be conscious of and annoyed by this behavior. Men tend to assume that everyone will ignore it. This causes problems in offices as well as in online communities, where women will complain about such behavior, and men will issue responses such as “toughen-up”, or “what’s the big deal?”, because this is how they cope with the problem. A female-friendly group addresses and tries to resolve these issues, while the average group ignores it until/unless the person does something heinous.

The sense of community fosters a protective behavior within that community.

If you do something awful to one woman in a women-only community, all will hear and know about it and you are ousted. Most of the time, this is first discussed and voted on by many group members. Many times the women’s group will even make an effort to explain the offense to the oblivious offender. But if the offender is still oblivious and/or offending, the offender is out. This is done to protect the interests and goals of the group. Many male-dominated online groups don’t run this way. Most if not all women’s groups run this way, whether online or off. This relates to the awareness and accountability mentioned before. It’s an essential element of all women-only groups and seems necessary for women-friendly groups to draw women.

Women’s groups generally have a few vocal and many silent members

The vocal few express their opinions and either gain support or do not gain support. The ones who gain support usually implicitly become spokespeople for the silent many. The silent many usually let the vocal few, with whom they agree, do the job of ousting, protecting the sense of community and publicly representing the silent many. The silent many support the vocal few. The community in turn supports and protects the rights and privileges of the silent many.

Why this happens is again a dynamic which is tangential to this article. But it seems that many women in group participation give either their silent support or rejection, speaking up only occasionally. Because of this behavior, if a communication problem arises in any type of group, whether women-only or not, and there are not a vocal few who will attempt to resolve it, the silent many will often silently leave. The silent many often don’t want to complain, for fear of having to deal with the additional frustration of the unaware/unconcerned “toughen-up”, or “what problem?” type of responses. For the silent many, it’s easier and less frustrating to just leave. I think it is important for groups that want to advertise themselves as being women-friendly to be aware of this pattern.

One of the challenges of any women-only/women-friendly group is encouraging the silent many to speak up. Many women deal with demeaning and discriminatory behavior so often in their lives that they are too emotionally exhausted to deal with even the possibility of an online onslaught of anonymous discriminatory and demeaning comments. Many women spend time observing online groups before deciding if they will participate, for this very reason. They want to ensure that they will not feel verbally attacked once speaking up and that their issues, comments and contributions will be heard and handled fairly.

Women generally do not arm themselves for battle during tech discussions

Women generally do not work things out through verbal battle. By the time they reach that the point of wanting to argue, they are already so offended that they are in pure self-defense mode. Women treat the discussion of tech issues like the discussion of many other issues. It’s not competitive and they wish to bi-directionally share information.

Many tech men envision a technical debate as a battle and celebrate the supposed victory, exhibiting classic ‘Alpha Male’ behavior. I have personally seen it so many times in my profession that I brace myself for it when discussing tech issues with new groups of men. So

many of them arm themselves with weapons of aggression, demeaning comments, and behavior which encourages more of a filibuster than a healthy debate. The supposed tech discussion becomes a test of verbal and emotional endurance, where whomever can argue the hardest and last the longest wins.

They can shake hands afterwards and congratulate each other over a 'good fight' after a technical debate. "I like the challenge of a good argument, which is why I do that", one male colleague explained to me. "I like a good technical debate too, but I don't want to feel verbally or emotionally abused afterward. Women don't fight for fun, they fight for personal issues." I explained to my male colleague.

Unfortunately, the anonymity offered by many public wikis and message boards encourages the worst behavior in people. Even moderated tech chat areas and comment boards are rife with elitist, demeaning comments encouraging 'the fight'. Some of it is due to oblivion, lack of knowledge that this is offensive to tech women. Some of it, unfortunately, is very intentional.

Apparently there are males online, in tech communities, who still believe that, like the cigar rooms of the Victorian Era, tech rooms should be male-only. Back then, the predominant purpose of smoking cigars in a common room was to have male-only space, and similarly today, the purpose of the demeaning and fight-provoking attempts is to maintain the male-only presence of some online tech spaces. I know for a fact this happens with intent in some online chat rooms and message boards. It is not simply an act of oblivion, but a concentrated, misogynistic effort between like-minded men to keep women out.

When I discuss this with people and we ask each other how this can be prevented, I feel overwhelmed. How do we stop any/all of the human behavior which prevents us from evolving further? I have no answer to this, but I am certain that if less of this behavior is tolerated online, we at least squeeze people who discriminate into their own, personal hidden online spaces. There is no reason why we need to be subjected to every single person's beliefs or comments in the name of the First Amendment[1]. We all have a right to remove from our lives anything and everything which holds us back in some way, even that which is subtly harmful or offensive. Web admins have a right to remove useless, demeaning, even subtly harmful comments in the best interest of an online community. The operative word here is "community", and the appropriate questions is: "Does your public comment space contribute to a community, or is it just an open toilet that everyone can vandalize and pollute?"

Did you know?

When it was illegal for women to publish writing during various times in history throughout various countries, women published their work under male pseudonyms. **Today, many tech women still use male pseudonyms** when posting to lists or publishing tech articles. The reasons are to have their work read without bias, and to avoid misogynistic 'hyper-scrutiny' of their work. I have experimented with this myself using a male pseudonym to post articles, and being told that the articles are informative, useful, great. Six months later I republish the exact same article, using a different title and a female pseudonym, and suddenly the article is horrible, technically incorrect, useless. It's a fascinating study. I would love to see some prominent male techs publish under female pseudonyms, and watch the responses.

Women find it awkward to brag about their writing accomplishments published under male pseudonyms. For this reason, most of this work never gets credited to the correct person, and is never acknowledged on resumes or during job interviews. "How do I explain to a male 'potential boss' why I have chosen to use a male pseudonym, without bringing up the whole discrimination issue?" is what one female tech friend asked me. I had no answer for her. I have also let my work published under male pseudonyms fall between the cracks, into oblivion, not knowing what else to do.

1 Specifically, the right to freedom of speech, see http://en.wikipedia.org/wiki/First_Amendment

To make an online community more women-friendly, try these suggestions:

1. Monitor the public comments. Treat the public comments interface much like the front door to your home. You don't simply leave it open for any idiot to waltz in. You can be selective regarding who comes in, and what they do once they're in.

   Useless comments get deleted as quickly as they appear. Any non-technical, offensive, destructive, or off-topic comment is removed. This gives a clear message about will and will not be tolerated. As useful comments accumulate, useless ones are much less likely to appear.

2. The technically correct but aggressive/demeaning/overly harsh comment gets returned to the sender, asking the person to re-word using constructive criticism. Sounds like overkill, but it's not. The "You're wrong, here's the right answer" type of response constitutes picking a battle that most women won't fight, or won't even bother dealing with.

3. Treat your online space like a community. The web admin should act is if they're on the board of chosen freeholders, voting on issues which affect themselves and the entire community. Don't just throw up the comment space and leave it abandoned for vandals and other jerks. Maintain it according to the rules by which you want everyone to abide, and stick by your decisions. Have accountability for comments. Create a space where open discussion happens as if it were in an educational surrounding, not a seedy bar.

4. Explicitly state that your site is women-friendly. Doing this will encourage the silent many to speak up. Kick out the jerks who don't want your online space to take this direction.

For the men who care: Tips for communicating with women in Tech environments, online and Face-to-Face

1. Tech women usually express great enthusiasm about their work. They do what they love, and they love what they do. When a woman gets enthusiastic about her work and shares that enthusiasm with you, **it has absolutely nothing to do with you, or sex**. I cannot tell you how often I have seen this. Some men mix up their incoming signals, and a women's enthusiasm at work somehow translates to someone flirting with them at a bar. I have no idea how this happens, but it's profoundly sad to see it happen again and again. If you're lacking something in your life, please do not look to your female tech colleague to fill that niche. Do not even presume her mind is there even if yours is not, because hers is not, and your signal indicator needs serious recalibration.

2. Leave your libido at the door. Please. Women tech colleagues want to be appreciated for their brains, their technical expertise, their contributions and accomplishments. **Tech women do not give a flying s**t about what their male colleagues think of their attire, their make-up or their body parts**. Believe me when I say this is true. Women may give you a polite response, but on the inside they are offended, seething, and considering whether or not to go to their attorney. They will ask other women in the office or field if they too suffer from this problem, building an alliance against men in their company who do this. And soon you will have a legal problem. Leave it at the door, pick it up on your way out. No one else wants it.

3. Some tech women dress up for work. **It is NEVER for you**. Many tech women wear clothing which makes them feel good. For some, comfort is paramount, if for example the tech female is crawling through the ceiling, moving dusty panels and running CAT5 cable. For other tech women who would not get their clothes ruined at work, they like to dress up. "It makes me feel confident. I look at myself in the mirror and I feel good." my female colleague told me. For tech women at work, feeling "good" does not mean "sexy", and it is not for you at all. It is entirely about self-confidence, self-encouragement, and giving one's self the extra strength to prove they know their

stuff in a technical environment. Note the emphasis on “self”: it is entirely for her, by her, and your reaction is entirely irrelevant.

I have heard males say horrible things in professional environments like “Well, you wore that dress, you do look great in it, that must be the reaction you wanted. Isn’t that why you wear that dress?” The answer is no, fool, get over yourself.

4. Tech women are generally open-minded about what is commonly called ‘guy humor’ and ‘guy socialization’. Guaranteed, many of them, myself included, have male friends with whom they hang out on a regular basis, so this is far from a foreign concept to tech women. Chances are, the tech women of your group would enjoy your jokes and would like to be invited out for beers, as long as points (1) through (3) above are met. I personally enjoy and share many of my own raunchy or lewd jokes if I feel safe around the people with whom I’m joking. I enjoy hanging out afterwards over a beer or two, or going out late with ‘the guys’ to a bar to welcome the ‘new guy’. These things could be fun for everyone if (1) through (3) are in order.

5. To the men who do not do any of this: Thank you so much. We notice, and greatly appreciate this. I have been fortunate to work with some excellent men in tech and I wanted to thank you and the many others for not being this way.

6. No, women are not perfect. This article doesn’t imply or suggest that women are close to prefect and men are far from it. I know there are female stereotypes not mentioned in this article, mostly because I personally don’t find them in tech environments. Your experience may vary. All of these points can be applied to both genders. But the fact that I was asked by several different sources to write this article proves that there is a recognized gender divide in many tech spaces. All of what I have posted is what I and others have observed and experienced. None of it is fiction.

7. Is someone making you feel uncomfortable? Speak up! If someone at work makes you feel uncomfortable, tell them so. If you feel discomfort coming from another person, and you think you’ve caused it inadvertently, say so. Make it clear and shove it out of the way as quickly as you can, so work can continue. This applies from/to men and women.

8. But isn’t creating a women-only group, and using terms like ‘male behavior’ reverse sexism? Doesn’t this defeat the very goal you wish to achieve? My response is no, not if these tools/verbiage are used to try to ultimately achieve equality. If it’s used for mudslinging, or through some act of elitist exclusion, yes, it is reverse sexism.

Credits: Many thank yous to Carla Schroder for sharing her infinite wisdom and encouragement. A huge thank you to all of the women at LinuxChix.org for your tireless support of the cause over the years. Thank you to DevChix.com for giving my wayward articles a very worthy home. Thank you to the many readers who have left constructive criticism and comments.

More writings by this author can be found at www.devchix.com.

Letters to The Python Papers

Python users from across the globe have written to the Python Papers to let us know what they're up to.

Krys Wilken

Krys wrote in to let us know about one of his blog posts which he felt expressed how much he thought of Python. We agreed! Krys' blog may be found at http://krys.ca/

Wow, it's been a while since I blogged. :(I've been busy obsessively coding a new reporting framework for work to replace a horribly designed/architected/coded/styled/everything ASP site. I have to say that:

1. setuptools completely rocks! Entry points are beautiful!
2. PasteScript completely rocks!
3. RuleDispatch *(which needs it's own site/page, btw)* is extremely handy in the right situations!

:D

With just these three libraries, I have created a completely extensible reporting framework with a very nice command-line interface and template generation to get new reports off the ground quickly. And it's only around 250 lines of code! :) (... so far, anyway. Not done yet.)

I also discovered inspect.getargspec recently and I am using it, and dependency injection, to add a couple bits of really nice elegance to the framework.

All this, combined with SQLAlchemy, makes generating reports in several formats from legacy (and badly designed) databases a walk in the park!

Thanks to learning these tools, I believe I have been able to elevate the level of beauty and quality of my code considerably.

Now, before anyone asks, the framework will very likely not be open-sourced, as I coded it at work and so it belongs to them.

That said, I just want to say a great big fat **Thank You to Phillip Eby, Ian Bicking, Michael Bayer** *(could not find a blog or homepage for you, Mike)*, and everyone that helped make the above libraries. Not only do they make my life easier, but using and studying them makes me a better programmer. And while I have derived great satisfaction from most of my coding projects, this is the first time in a long while that I have actually been euphoric about it! This stuff is just cool! :D I mean, is this what Lisp programmers feel all the time? ;)

Interestingly, it's not that I'm proud of myself, it's that I am completely blown away by what Python and these extremely powerful tools (and other like them) let me accomplish. I try not to be a fanboy, but you know, sometimes it's really hard! :)

Michael Ang

After our call to Pythonistas in a variety of countries, Michael wrote in to let us know a little about how Python is used in Singapore...

"I am from Singapore, and a Python user... Have been using Python for about 7-8 years, and my other core programming languages are Delphi, Clipper (xHarbour), and many others. The only sad thing is Python is not popular here, our academics emphasize very much to Microsoft, so graduates tend to be more familiar to VB and what not. "

Anyway, I just needed to express that. If you have not checked out what these libraries can do, I highly recommend it. They are all obviously designed to scratch and itch and, at least for me, they broadened the way I think about program design.

Just to finish this post off, I am curious if anyone else has had this kind of experience with a Python library/tool/language feature. If so, I'd love to hear about them! This might even provide some nice material for the Python Advocacy list.

Anyway, thanks again guys! You rock!

Peter Williams

Peter is a subscriber to the Melbourne Python User's Group[2] mailing list, and works for the New South Wales (Australia) Rural Doctors Network. Thanks to him and his organisation for allowing us to reproduce this email...

"We are developing our operational database systems in-house almost entirely using open source software products. We use Postgresql with a Python front-end. Our database includes a schema which we call "metadata" – basically a database of how our data structures are designed and how they are to be presented by the python front-end. Our users are located in several offices around NSW, and we use a secure shell for remote access (head office is here in Newcastle). Our users also have a variety of platforms – Windows, Mac, Linux, so this combination of Postgresql + Python works very well for us. When a user logs in, the metadata schema is queried to determine how to present the various "screens" of data for the user. If we want to change the data structures, we change the records in the metadata schema, and then run a "rebuild" program, which drops data, then drops database structures, then reads the new metadata, creates new database structures, and then imports data that was previously dropped. The rebuild takes about 15 minutes, and we usually schedule rebuilds at lunchtime. Our system has several major functions. Firstly, our "core module" is a CRM module. To this we have added a number of facilities to help users communicate (calendars, task lists, resource booking system, alerts, procedures, meeting agendas and schedules, staff whereabouts etc etc). Then there are modules that are specific

Brian Blais

Brian Blais works as an Associate Professor at Bryant University. He has been corresponding with The Python Papers about his insights on the use of Python in Universities, and was happy for us to include this message...

"I have been using Python for only about 1.5 years, coming primarily from a Matlab+C background. I currently use it both for teaching and research, and it has transformed the way I work. I've used it for student projects in Astronomy, Physics, Artificial Intelligence, Robotics, Computational Neuroscience, and Meteorology. For research, I do a lot of numerical simulations (mostly neurons), and I've transferred all of my heavy-hitting Matlab+C code into Python+Pyrex."

Flavio Codeco Coelho

Flavio has been of help to The Python Papers reviewing some of our academic contributions, and also keeping us up-to-date with regards to University research.

"I am a Brazilian researcher in the field of Biomathematics and I have been involved with Python for many years now, both in the development of scientific software (http://epigrass.sourceforge.net and http://modelbuilder.sourceforge.net/) and in training young scientists to use Python for their computational needs. I am currently finishing a book about Python in Science, with should come out this year, in portuguese (Translation offers welcome!). I also keep a blog about Python in Science http://pyinsci.blogspot.com"

2. http://wiki.python.org/moin/MelbournePUG

to certain project areas eg locum service module, general practitioner workforce module. We plan to add more modules as we continue to develop the system. We also have a generic data collection instrument facility staff can use this to define questions, surveys (or data collection instruments), and then enter data from survey respondents or applicants. Some of these collect data directly from stakeholders via a php web interface, directly into the database. Triggers then create alerts for appropriate staff members to take action. We use ReportLab for our reporting requirements, and all reports are created as pdf documents which are then emailed to the user (no printer driver fuss!). ”(Object Oriented) Python Rant

Alex Nelson

Alex Nelson, originally Wednesday, August 1, 2007 on his blog “Object Oriented Kool Aid”.[3] *Alex is a sophomore physics/math double major at the University of California at Davis. His research interests include quantum gravity, foundational issues in quantum mechanics, and general relativity. In his spare time, Alex studies file systems, and works on the Brainix operating system.*

Python is the way of the future...like Zeppelins and Autogyros. It's a lot more natural programming in Python than it is in Perl, at least for Object Oriented Programming. (Note that there is an interesting series called the Python Papers that is a good read!) [*editor's note – we did not make him say this!*]

When I first learned it, I hated it because it wasn't like C/C++/D/Java at all! However, I gave it high marks for having the `lambda` anonymous functions. It made me reminisce about the bad old days of LISP and SCHEME and *The Structure and Interpretation of Programming*.

Now, my tune has changed **completely** since I'm working on an operating system (Brainix). I recognize the value of having scripts, despite the fact that Brainix is not mature enough to run a simple shell. It's the benefit of platform independence at the cost of performance.

The problem is that there **are no good scripting languages!** Perl is esoteric as death, and BASH is not all that better...don't get me wrong, BASH is fabulous as a shell but terrible as a scripting language. Python is perhaps the best scripting language out there, and that's not saying all that much.

One particular problem that I have is, for perl I can do things like:

```
@files = `ls`;
for $file in @files
{
#do stuff!
}
```

However, for Python, having this neat scripting feature is changed! In the immortal words of "Grandpa" Simpson "I was with it once...then it changed into something horrible and scary." After a bit of study, I figured out that there is an `OS` module where I can use the `os.system()` method to invoke shell commands. Now I could write something like:

```
for filename in os.command("ls"):
# do stuff translated into python!
```

Perhaps a more disturbing difference for my inner C/C++/D/Java programmer is the lack of curly brackets. I got over this by using Python in a more functional manner that would make John Armstrong proud.

Finally I fought my fears and started programming in an object oriented manner. Object oriented programming is the way of my people! But this sort of object orientedness seemed odd. The `self` pointer (I assume, I thought it to be a python parallel to the `this` pointer at first) is the first argument in every method, which was bizarre. It reminded me of Phil's Object

3 http://pqnelson.blogspot.com/2007/08/object-oriented-python-rant.html

Oriented ANSI C.

Perhaps what the open source movement needs is a good shell that's also a good scripting language...one that's open source, object oriented, and is an affront to the Windoze PowerShell. BeanShell is a possibility, but I think there is hope for python yet.

Well, this isn't much of a first post, but that's all I have to rant about so far about Python.

The Longest Common Substring and Sentence Modification

Tennessee Leeuwenburg

I had this problem. Some sentence (A) was modified somehow. Given the new sentence (B), identify what was added and what was removed, if anything? To solve this problem required a tree structure for the edits, a method for extracting the longest common substring and a little ingenuity.

First, let's look at why one might want to do such a thing. The example for which this code was developed requires a bit of expert knowledge, so what follows is hopefully a more generally applicable idea.

Supposing you're a Wikipedia reviewer and someone has just submitted a modification to an article. This article may be quite long, perhaps a couple of pages, while the edits may be quite short. In a program like Microsoft Office or Open Office, change-tracking may be employed to follow changes to a document by tracking each character modification to the original as it happens. I'm not an expert on the editing interface to Wikipedia, but I don't believe it tracks changes. So the reviewer has to manually compare the new, edited version with the original. They must rely on their ability to *recognise* changes between the two versions.

Wouldn't it be great if you could do this automatically?

For some applications, there's obviously a great tool out there which does this already – diff[4]. However, diff works on a line-by-line basis and isn't that easy to integrate into a Python application. The utility outlined here works on a word-by-word basis, allowing differences between documents to be highlighted very specifically. Moreover, the basis for tokenising can be overridden, allowing character-level differences to be shown if desired.

The remainder of this article describes the means by which two documents may be compared and their edits identified and then covers the result of a public code review on the Melbourne Python User's Group mailing list.

Comparing Two Documents

Supposing I have the following two sentences:

“I was walking down the street one morning” and
“I was walking down the street early one morning”.

The modification here is fairly clear – the word “early” was inserted between “street” and “one”. Humans are pretty good at spotting this kind of thing and it's also an easy example.

There are a number of possible algorithms that can be imagined for identifying edits mechanistically. For example, the sentences could be considered front-first until the words were different (e.g. “I was walking down the street...”). Where do they merge again? In this case, it's the word immediately afterwards. However, supposing it wasn't that one. Maybe it's the word after that? Looks like we'll have to check every single word in the next sentence for similarity.

Come to think of it, how do we know only one word was changed?

“I was walking down a darkened street one morning”
“I was walking down the street one fearful step at a time, one morning”

Hardly poetry, but it demonstrates the point. The best match for the sentence fragment “one

4 http://en.wikipedia.org/wiki/Diff

morning” is not at the first occurrence of “one”. It happens to be at the end of the sentence, so maybe a reverse-search could identify it. But that's not what makes it the best match. What makes it the best match is that “one morning” is a longer contiguous matching block than simply “one”.

It is here that it is apparent that our eventual algorithm will have to prioritise longer matching blocks of text over shorter matching blocks of text.

This insight can be used to design an algorithm for identifying the smallest possible edit required to get from A to B. The smallest possible edit is exactly the complement of the longest possible match.

Only insertions and deletions are considered here. This means that “One morning I walked down the street” and “I walked down the street one morning” are quite dissimilar. While this could be represented by a single **move**, in terms of **insertions** and **deletions** it is several operations away. For the purpose to which the algorithm was put, that was the appropriate behaviour. Applications for, and implementations of, a move-based analysis of two documents could be fruitful areas for further investigation.

On that basis then, the words of both documents are guaranteed to share the same order. If a matching text block occurs in both (A) and (B), only the words before the match in (A) are potentially similar to the words before the match in (B).

This allows us to use a divide-and-conquer approach to the problem.

If the longest common substring can be identified, a sentence may be broken into three parts – that preceeding the match, the match itself and that following the match.

A: “I remember that **I was walking down** a darkened street one morning”
B: “It seemed as though **I was walking down** the street one fearful step at a time, one morning”

This can now be broken down into a shared longest common substring, a pair of prefixes and a pair of suffixes.

The pair of prefixes, “I remember that” and “It seems as though” have no common substrings. The progression from A to B therefore represents a deletion of the first prefix and the insertion of the second, followed by the common substring. The pair of suffices “a darkened street one morning” and “the street one fearful step at a time, one morning” have further common substrings, so the analysis can continue.

One representation of the full tree for this example is shown in figure one.

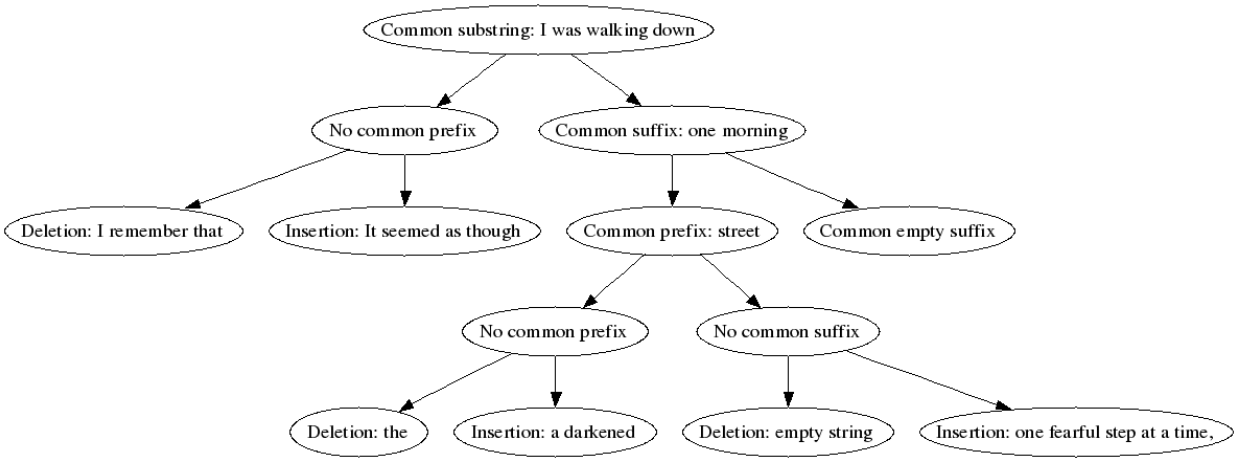


Figure 1: Final tree structure showing edits

The tree structure can then be used to extract either sentence, or show the full deletion and insertion sequence.

The Code

The full code for the algorithm can be found immediately below. Following this section will be a discussion of the code, including modifications which were made as a result of the peer review process, recommendations which were not pursued and other finishing comments.

```
import re

class SentenceComparison:
    '''
        This class represents the comparison of one sentence with another. It produces
        a tree showing the differences. From this can be reconstructed the original
        sentences, showing where the sentences differ.

        Each word in a sentence is atomic.
    '''

    def __init__(self, string1, string2, depth=0, tokenize=str.split):
        '''
            string1: the first sentence
            string2: the second sentence

            If unicode strings are to be used, the tokenize function
            will need to be overridden to be unicode.split
        '''

        DEBUG = False

        if DEBUG: print "__init__: depth %s called with %s : %s" % (depth, string1, string2)

        self.tokenize = tokenize

        self.string1 = ' '.join(tokenize(string1))
        self.string2 = ' '.join(tokenize(string2))

        self.lcs = ""
        self.lTree = None
        self.rTree = None
        self.depth = depth

        self._buildTree()

    def _buildTree(self):
        '''
            Based on string1 and string2, build up the (prefix, common, suffix) tree
            structure
        '''

        lTree = None
        rTree = None

        string1 = self.string1
        string2 = self.string2

        DEBUG = False
        if DEBUG: print "_buildTree depth %s: (%s : %s) " % (self.depth, string1, string2),

        if string1 == "":
            if DEBUG: print 'a'
            lTree = ""
            rTree = self.string2
            self.lcs = ""

        elif string2 == "":
            if DEBUG: print 'b'
            lTree = self.string1
            rTree = None
            self.lcs = ""
```

```
        else: #Both strings contain text
            self.lcs = self.LCS(string1, string2, tokenize=self.tokenize).strip()

            if self.lcs == '':
                lTree = string1
                rTree = string2

            else:

                tuple1 = string1.split(self.lcs, 1)
                if len(tuple1) == 2:
                    prefix1, suffix1 = tuple1
                else:
                    [prefix1] = tuple1
                    suffix1 = ''

                tuple2 = string2.split(self.lcs, 1)
                if len(tuple2) == 2:
                    prefix2, suffix2 = tuple2
                else:
                    [prefix2] = tuple2
                    suffix2 = ''

                lTree = SentenceComparison(prefix1, prefix2, depth=self.depth + 1)
                rTree = SentenceComparison(suffix1, suffix2, depth=self.depth + 1)

        self.lTree = lTree
        self.rTree = rTree

    def LCS(self, string1, string2, tokenize=str.split):
        '''
            Based on string1 and string2, returns the longest
            common substring, on a word-by-word basis using
            a word-matching regular expression.
        '''

        words1 = tokenize(string1)
        words2 = tokenize(string2)

        m = len(words1)
        n = len(words2)

        lengths = [[0] * (n+1) for i in xrange(m+1)]
        LCS = []
        longest = 0

        for i in xrange(m):
            for j in xrange(n):
                if words1[i] == words2[j]:
                    v = lengths[i][j] + 1
                    lengths[i+1][j+1] = v

                    if v > longest: longest = v
                    if v == longest: LCS = words1[i - v+1:i+1]

        return ' '.join(LCS)


    def lString(self):
        '''
            Print out the left string, noting additions and removals
        '''

        DEBUG = False
        if DEBUG: print "\nlString: (%s : %s)" % (self.string1, self.string2),

        lTree = self.lTree
        rTree = self.rTree
        lcs = self.lcs

        myString = ''

        if lTree is not None:
            if isinstance(lTree, basestring):
                if lTree is not "":
```

```
                    myString +=  " (added %s) " %  (lTree)
            else:
                myString += lTree.lString()
        else:
            if DEBUG: print 'lTree is None'

        if lcs != "":
            myString = myString + lcs

        if rTree is not None:
            if isinstance(rTree, basestring):
                if rTree is not "":
                    myString  += " [removed %s] " % (rTree)
            else:
                myString += rTree.lString()

        return myString


if __name__ == "__main__":

    string1 = "these two strings"
    string2 = "are completely different"

    a = SentenceComparison(string1, string2)
    print a.lString()
  print "See LCSTest.py for further tests"
```

Comments and Responses

In this section, I include the comments of other developers on the approach taken. For the sake of space, I have not included the full code both before and after incorporating the suggestions made by those on the list, but rather have simply described the changes made as a result of each individuals' comments.

Maurice Ling:

1. How are the words identified? By whitespaces? If so, then there is a false removal (substitution) in this case: **original:** "**Tom ate an apple"**. *new:* "*Tom ate an apple and an orange"*.

2. Hyphenations etc? For example, "**Tom is twenty-three years old this year**" and "*Tom is twenty three years old this year*".

Tennessee:

At the moment, str.split is used, which will result in a whitespace based split. This does indeed result in some undesirable behaviour due to punctuation. In practical application, this is of limited impact. Indeed, depending on the requirements of each case, the response to punctuation might be different. In some cases, perhaps especially that of program code or legal writing, punctuation can be critical. In other cases, it can be virtually ignored.

In response to this point and others, it is possible to pass a tokenizer method in which allows people to change the word (or token) identification method.

John Machin:

Capitalisation is another problem: original: "Envy and pride are ..." new: "Sloth, envy and pride are ..."

Comments say "words are atomic": what about typos? stuff cheesw?

At the Python level -- based on [possibly incorrect] recollections from reading it yesterday; detailed dissection later :-)

1. tokens produced by str.split() don't need str.strip() applied to them

2. blank lines in unexpected places e.g. before else:

3. "if not thing is None" -- syntactically correct but stylistically chundrous IMHO; what's wrong with "if thing is not None"?

4. put in comments that explain your tree structure, or at the very least position the tree creating method(s) before the tree-examining method(s) -- save gentle readers the need to nut out the meaning of: "node is None", "node == "", isinstance(node, str) # what about unicode? node is none of the above

5. Testing/example architecture could be a bit more robust than a collection of commented pairs of sentences down the end.

Tennessee:

More good points. Capitalisation, as with punctuation, may be very relevant or not be relevant at all.

Spotting typos, I feel, is a use case for the algorithm. It is probably a good thing that typos will cause an edit to show. However, should somebody wish it, I can imagine one approach. Somelike like the Levenshtein distance [5] could be used to determine if an edit is likely to be a typo or a genuine change of word. In this case though, it's beyond the requirements.

The bulk of the remainder of the comments relate to coding standards (i.e. PEP-8[6]), poor idiom and just a little outright confusing code. :) The first two of those criticism were hopefully addressed. A few differences may remain, but several were addressed.

To explain the aspect of the code which caused such confusion, a node may either be an intermediate node or it may be a leaf node. In this case, leaf nodes are identified by being of type basestring. It would almost certainly be preferable if designing a public API to explicitly

5 http://www.cut-the-knot.org/do_you_know/Strings.shtml

6 http://www.python.org/dev/peps/pep-0008/

identify leaf nodes and retrieve the string from them. For reasons of time, this suggestion was not adopted, but should be kept in mind for any future work.

The original code contained a number of input/output pairs in the same file as the SentenceComparison class. This is, to say the least, not industry-leading practise. It did, however, get the job done at the time of initial writing. In order to explore testing and also to be able to present a reasonable approach to good coding in this article, an additional file containing unit tests was created. This testing code was not reviewed, but is included at the end of this article for readers' reference purposes.

Anthony Briggs:

A few other points - these are stylistic though, which I'm not sure is what you want, but anyway:

```
if lcs != "":
    myString = myString + lcs
```

is a no-op as far as I can tell. Since you only use it the once, you probably also don't need the 'lcs = self.lcs' part either.

Tennessee: *I checked this – turns out it's needed after all.*

You've also got a couple of places where you're comparing the left side of the tree and then the right side of the tree. For example,

```
if not lTree is None:
    if isinstance(lTree, str):
        if lTree is not "":
            myString +=  " (added %s) " %  (lTree)
    else:
        myString += lTree.lString()
else:
    if DEBUG: print 'lTree is None'
```

and the other rTree one could become something like:

```
myString += self.parseTree(lTree, 'added')
myString += self.parseTree(rTree, 'removed')
```

Similarly for the other tree building/exploring functions (lines 109, 116).

Tennessee: *Yes, I can see the good sense in this. Another one for the 'should-do' list. It would greatly help in making the code easier to scan.*

Other picky code style type things: On 76 + 77, you set lTree and rTree, even though all three branches set them anyway.

You seem to be using a fair few placeholder methods, and then not using them, eg. string1 and string2 on lines 79 and 80.

When you're comparing string1 and string2, you might benefit (in terms of clarity of code) from returning early. eg.

```
if string1 == "":
    self.lTree = ""
    self.rTree = self.string2
    self.lcs = ""
    return

if string2 == "":
```

```
...
```

And you seem to be running if statements onto one line, which I find makes things harder to read, eg.

```
if v > longest:longest = v
if v == longest: LCS = words1[i – v+1:i+1]
```

would (IMO) be better as:

```
if v > longest:
    longest = v
    LCS = words1[i - v+1:i+1]
```

Tennessee: *I checked PEP-8 and while one-line ifs are discouraged, I feel that I have just enough wiggle room to stick with my preferred style, which is to use one line if statements liberally. I must say though, I believe the majority of people prefer to use multi-line if statements in all cases – so readers, probably best to follow Anthony's suggestion. For this specific example, also, I think the multi-line statement makes the logic clearer.*

Ryan Kelly:

(replying to an email of Tennessee's)

T> 4) isinstance(node, str) -- indeed, what about unicode? In Python 2.5, is a unicode string a T> str? I'll have to research this to make sure.

From memory, the 'proper' way to do this is to compare with basestring, although everytime I use it I cringe slightly because it just doesn't read right:

```
isinstance(node,basestring)
```

Tennessee: *You can see that I have incorporated the use of basestring into the code. I would agree with Ryan that it reads clumsily to use basestring. Unicode generally is pretty tricky to deal with, so it's hard to suggest something obvious for Python to do about it.*

T> 5) Testing. I'm not familiar with unit testing frameworks. The best thing would probably T> be to identify some kind of preferred testing framework and write a better set of formal T> tests. Any suggestions?

I've had good experiences with the built-in unittest module, particularly using it via setuptools `python setup.py test` command. I've got simple needs but it's saved by bacon a few times :-)

Tennessee: *In the end this is what was chosen, for two major reasons. One was the recommendation from Anthony. The second is the major advantage that the module comes installed with Python by default. This means that the code used below will run anywhere, which is particularly appropriate for a magazine article.*

In Conclusion

This article has now presented the problem (loosely specified), an algorithm to identify edits between two documents and made use of code review to improve on the system. Exposing ones code is always somewhat difficult. Indeed, it is rather with trepidation that I include my code in this magazine. The process of code review through the mailing list, however, I found to be entirely positive. If others are working on code and have access to a peer-group, I think that using them as a sounding board can be really good for your code, as well as giving you pointers to technologies which you might not be aware of or simply never fully explored. In my case, this was what tipped me over the edge to explore the unittest module.

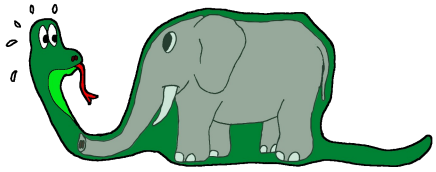
Testing Code

```
import unittest
from LCS import SentenceComparison

class LCSTest(unittest.TestCase):

    testsMappings = [
                ("Wind east to northeasterly tending southerly",
                "Wind northeasterly tending southerly",
                "Wind (added east to) northeasterly tending southerly"),

                ("Wind northeasterly tending southerly",
                "Wind east to northeasterly tending southerly",
                "Wind [removed east to] northeasterly tending southerly"),

                ("Wind east to northeasterly tending southerly",
                 "Wind northeasterly tending southerly around midday then increasing to 20 to 25
knots",
                 "Wind (added east to) northeasterly tending southerly [removed around midday
then increasing to 20 to 25 knots] "),

                ('''<p>
                                                    Sunny. Winds north to northwesterly at up to 15
km/h tending northeasterly at 15 to 30 km/h around midday. Temperatures in the mid 20s during the
day.
                                            </p>''',
                 '''<p>
                                                    Sunny. Winds north to northwesterly at up to 15
km/h tending northeasterly at 15 to 30 km/h around midday. Temperatures in the mid 20s during the
day. Green eggs and ham.
                                            </p>''',
                 "<p> Sunny. Winds north to northwesterly at up to 15 km/h tending northeasterly
at 15 to 30 km/h around midday. Temperatures in the mid 20s during the day. [removed Green eggs
and ham.] </p>"),

                ("Wind",
                 "Wind east to",
                 "Wind [removed east to] "),

                ("", "", ""),
                ("Winds northeast to northwest", "", " (added Winds northeast to northwest) "),
                ("", "Winds northeast to northwest", " [removed Winds northeast to northwest] ")
            ]

    def testInit(self):

        lcs = SentenceComparison("These two strings", "are completely different")
        assert lcs is not None

    def testLCS(self):

        lcsTests = [
                    ("", ""),
                    ("", "a"),
                    ("a", ""),
                    ("these two strings", "are completely different"),
                    ("these strings are the same", "these strings are the same")
                    ]

        for (string1, string2) in lcsTests:
            sc = SentenceComparison(string1, string2)
            substring = sc.LCS(string1, string2)
            assert substring is not None

    def testLString(self):

        for (string1, string2, output) in self.testsMappings:
            a = SentenceComparison(string1, string2)

            try:
                self.assertEqual(a.lString(), output)

            except:
```

```
                error = "\n<string1>: <%s> \n<string2>: <%s>\n     did not produce expected output
\n<%s> \
                         \n     but rather \n<%s>" \
                        % (string1, string2, output, a.lString())

                self.fail(error)

def suite():
    suite = unittest.TestSuite()
    suite.addTest(LCSTest("testInit"))
    suite.addTest(LCSTest("testLString"))
    suite.addTest(LCSTest("testLCS"))

if __name__ == "__main__":
    unittest.main()
```

Pyphant – A Python Framework for Modelling Reusable Information Processing Tasks

Klaus Zimmermann, Lorenz Quack and Andreas W. Liehr



We are presenting the Python framework “Pyphant” for the creation and application of information flow models. The central idea of this approach is to encapsulate each data processing step in one unit which we call a **worker**. A worker receives input via **sockets** and provides the results of its data processing via **plugs**. These can be connected to other workers' sockets. The resulting directed graph is called a **recipe**. Classes for these objects comprise the Pyphant core. To implement the actual processing steps, Pyphant relies on third-party plug-ins which extend the basic worker class and can be distributed as Python eggs. On top of the core, Pyphant offers an information exchange layer which facilitates the interoperability of the workers, using Numpy objects. A third layer comprises textual and graphical user interfaces. The former allows for the batch processing of data and the latter allows for the interactive construction of recipes.

This paper discusses the Pyphant framework and presents an example recipe for determining the length scale of aggregated polymeric phases, building an amphiphilic conetwork from an Atomic Force Microscopy (AFM) phase mode image.

[*This paper was originally presented at Europython 2006 and has been updated for this publication. Full acknowledgements are included at the end of this article. -Ed*]

1. Introduction

Working as a computer scientist in an interdisciplinary scientific community often means adapting a previously developed data processing algorithm to the very special context of a new project. An example might be that of image processing[7]. Consider that you have previously developed an algorithm, which determines the particle size distribution of a certain blend of materials on the basis of an Atomic Force Microscopy (AFM) measurement. Given the measurement of a different material, it is likely that you would have to apply different processing steps in order to match the characteristics of the new sample. If you consider a programming environment which assists the adaptation of this data analysis algorithm, you will very quickly consider a flow-based programming paradigm. This approach was invented in the late sixties[8] and has become established. This can be seen from the variety of commercial and open-source tools applying flow-based programming in the context of visual programming languages[9]. As regards data analysis, several flow-based environments have been implemented in Python, including ViPEr[10] and the Modular Toolkit for Data Processing

7. John C. Russ. *The Image Processing Handbook*. CRC Press, Boca Raton, 4 edition, 2002.
8. John Paul Morrison. *Flow-based programming: A New Approach to Application Development*. VNR computer library. Van Nostrand Reinhold, New York, 1994 http://www.jpaulmorrison.com/fbp/index.shtml.
9. Wikipedia. *Visual programming language*. http://en.wikipedia.org/wiki/Visual_programming_language
10. Michel F. Sanner, Daniel Stoffler, and Arthur J. Olson. *ViPEr a Visual Programming Environment for Python*. In *10th International Python Conference*, February 2002. http://www.scripps.edu/~sanner/html/papers/IPC02.pdf

(MDP)[11]. It has also been demonstrated that Python is well-suited to integrate several different software tools, e.g., for computation and visualization into a consistent data analysis environment[12].

Inspired by these approaches and having in mind that quite different data analysis tools, ranging from standard algorithms of statistics or image processing up to specialized tools developed in the context of materials research[13] [14] [15], have to be included into a consistent visual programming environment, we started to think about the Pyphant framework. A major prerequisite was that the resulting environment should be suitable not only for the creative work of a specialized scientist but also for standardized data processing in a daily laboratory routine or a large-scale data analysis campaign, computed in a grid computing environment. An attempt to balance these needs resulted in the Pyphant framework, enabling the fast integration of software modules into so-called **workers**, which receive input data via **sockets** and provide their cached results via **plugs**. The data analysis algorithms are composed as directed graphs within the GUI **wxPyphant**[16]. The interactive evaluation of the algorithm is established using an extensible set of **visualisers**. Finally, the algorithm and its intermediary result can be saved in the Hierarchical Data Format HDF5[17]. The resulting file is also the basis for Command Line Interfaces (CLI), which can be tailored as Python scripts.

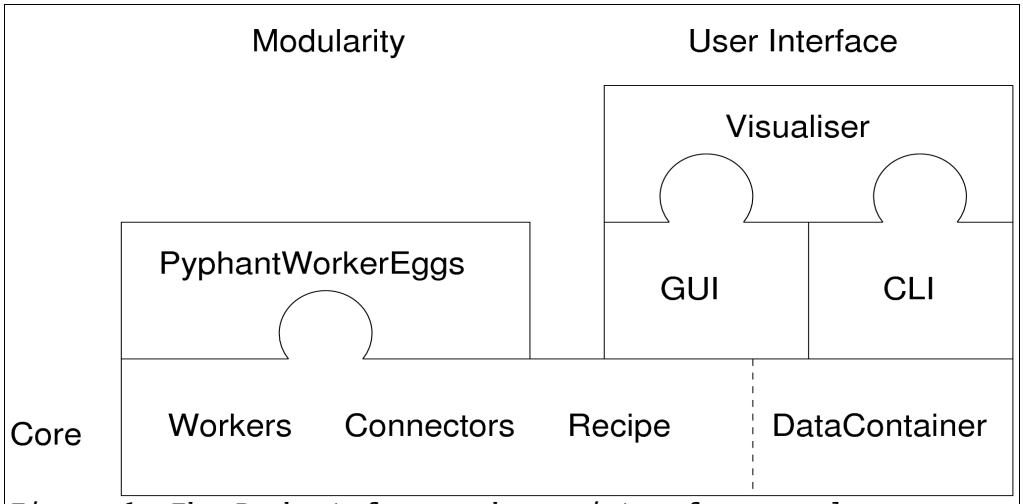


Figure 1. The Pyphant framework consists of a core layer comprising worker, connector, recipe and DataContainer objects. Workers for specialised data processing tasks can be provided by PyphantWorkerEggs. User interfaces are provided in the form of the GUI wxPyphant and a command-line interface for the individual recipe.

11. Pietro Berkes and Tiziano Zito. *Modular toolkit for Data Processing (MDP)*. http://mdp- toolkit.sourceforge.net, 2006.
12. M. F. Sanner, B. S. Duncan, C. J. Carrillo, and A. J. Olson. *Integrating Computation and Visualization for Biomolecular Analysis: An Example Using Python and AVS*. In *Proc. Pacific Symposium in Biocomputing '99*, pp 401-412, 1999.
13. J. Honerkamp and J. Weese. *A nonlinear regularization method for the calculation of relaxation spectra*. Rheologica Acta, 32(65):73, 1993.
14. T. Roths, M. Marth, J. Weese, and J. Honerkamp. *A generalized regularization method for nonlinear ill- posed problems enhanced for nonlinear regularization terms*. Computer Physics Communication, 139:279-296, 2001.
15. M. Bohnert, R. Walther, T. Roths, and J. Honerkamp. *A Monte Carlo-based model for steady-state diffuse reflectance spectrometry in human skin: estimation of carbon monoxide concentration in livor mortis*. Int J Legal Med, 119:355-362, 2005.
16. Servicegroup Scientific Information Processing, http://pyphant.sourceforge.net
17. The HDF Group (THG). *Hdf5 (hierarchical data format 5) software library and utilities*. http://hdf.ncsa.uiuc.edu/HDF5, 2006.

This paper begins with an overview of the Pyphant framework and continues with a real-life example demonstrating the estimation of the length scale of a phase-separated polymer blend.

2. Framework

Pyphant is a layered, plugin-based framework suitable for the modelling and execution of a wide range of information processing tasks. It is built on the idea that many computing algorithms can be structured into a graph of distinct steps. In Pyphant those steps are represented by so-called workers, which also form the nodes in the directed graph. Such an algorithm is called a recipe, following the famous textbook **Numerical Recipes**[18]. In this context the development of a data analysis algorithm can be pictured as the composition of a meal from various available ingredients.

Fig. 1 shows an overview of the structure of the Pyphant framework. At its base we find the core. Apart from the workers and the recipe, we have the connectors which are used to model the edges of our graph and which are usually members of the workers. Pyphant's core is completed by the DataContainer class. While the most basic incarnation of a Pyphant application does not impose any restriction on the data format exchanged amongst workers, we added this container format to enhance the interoperability of the various workers.

Above the core we find the user interface layer (UI-layer), which comes in two flavours. We have implemented a simple GUI called wxPyphant which realises the visual programming paradigm. Pyphant also facilitates the easy creation of a command-line interface (CLI) for a specific recipe. This feature is very useful for a daily laboratory routine or the analysis of large data sets, e.g. if large scale data mining is performed in a grid computing environment.

18. William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes in C*. The Art of Scientific Computing. Cambridge University Press, Cambridge, 2 edition, 1996.

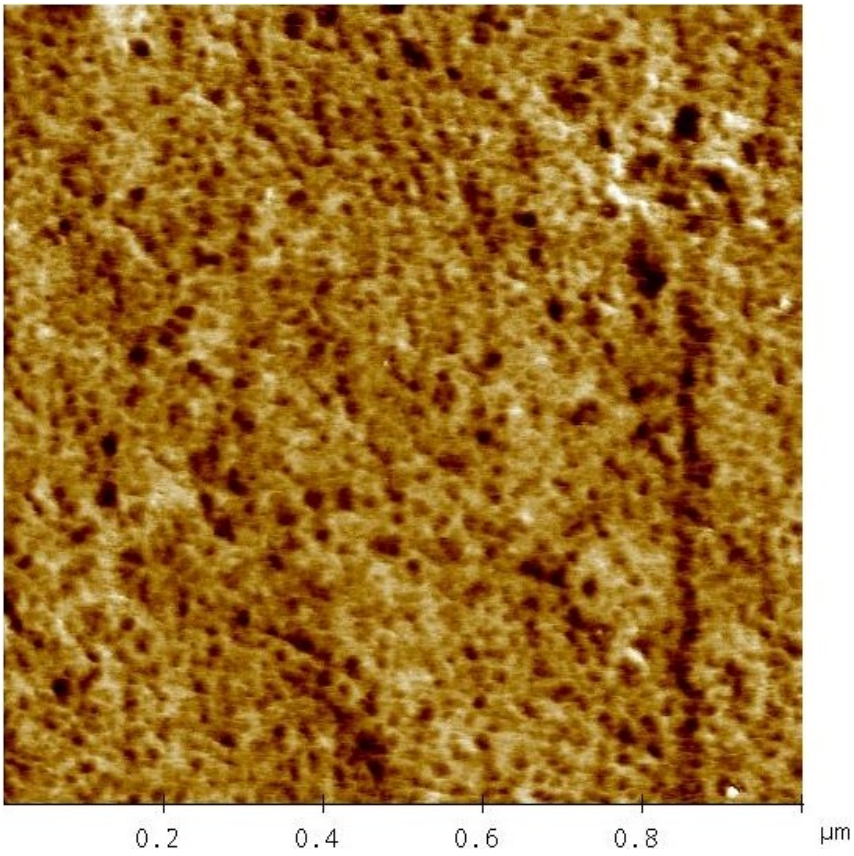


Figure 2. Atomic Force Microscopy (AFM) phase mode image1,14of an amphiphilic poly(2-hydroxyethyl acrylate)-l-poly(dimethylsolixane) (PHEA-l-PDMS) conetwork with 23 wt% PDMS. PHEA shows light and PDMS dark.

That's all for the Pyphant framework. However, it would not be useful if it wasn't for the plugins which do the actual work. The most important kind of plugin is the worker plugin. For now it suffices to notice that workers are bundled in PyphantWorkerEggs. Another kind of plugin is the visualisation plugin which is used to visualise data in a suitable way.

Now that you have a rough idea of Pyphant, let's start with a real-life example of a Pyphant application.

3. Image Processing Example

In this real-life example we will explain the steps needed to estimate the width distribution of an aggregated polymer phase from an AFM phase mode image. The example starts with loading the primary data, pre-processing the data and finally determining the size of the detected features. A possible evaluation step is also discussed.

Fig. 2 shows an AFM phase mode image of an amphiphilic poly(2-hydroxyethyl acrylate)-*l*-poly(dimethylsolixane) (PHEA-*l*-PDMS) conetwork with 23 wt% PDMS[19]. In this visualization

19. Nico Bruns, Jonas Scherble, Laura Hartmann, Ralf Thomann, Bela Ian, Rolf Mülhaupt and Joerg C. Tiller. *Nanophase Separated Amphiphilic Conetwork Coatings and Membranes*. Macromolecules 38 pp 2431-2438, 2005.

the PHEA and PDMS show light and dark respectively. The task is to determine the with of the PDMS phase.

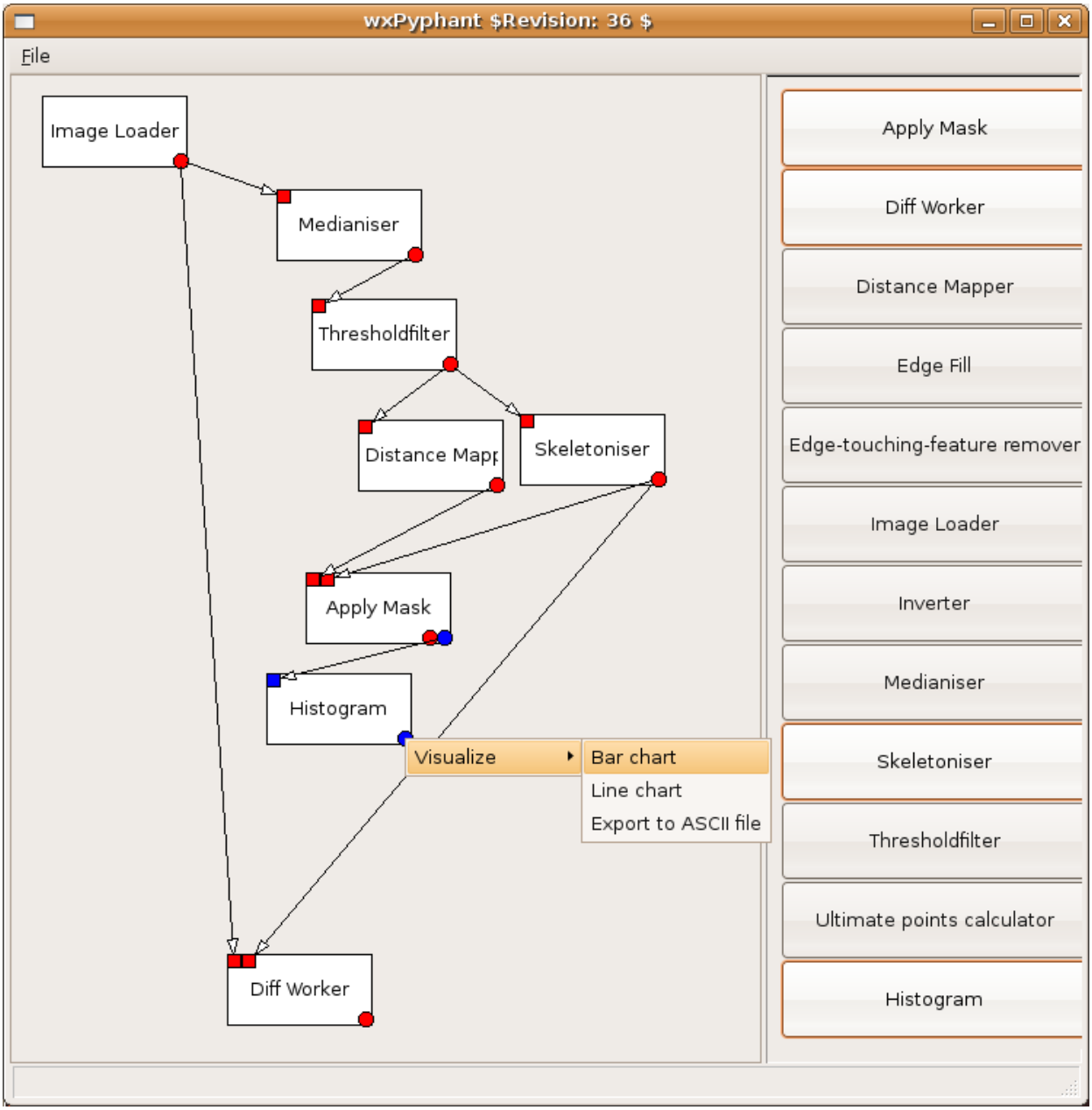


Figure 3. Pyphant recipe composed in the wxPyphant GUI. Workers are visualised as white boxes with sockets placed in their upper-left corner and available plugs placed in their lower-right corner. By right-clicking a plug, a context menu with visualisation plugins is provided.

The complete Pyphant recipe is depicted as a snapshot of the GUI in Fig. 3. On the right-hand side of the GUI, the toolbox of available workers is visualised. Each worker can be placed by drag-and-drop on the canvas. The individual workers are clearly visible as white boxes which are connected by arrows pointing from the plug of one worker to the socket of another worker. The colour of the connectors indicate different types of DataContainer. Red indicates a FieldContainer, while blue denotes a SampleContainer.

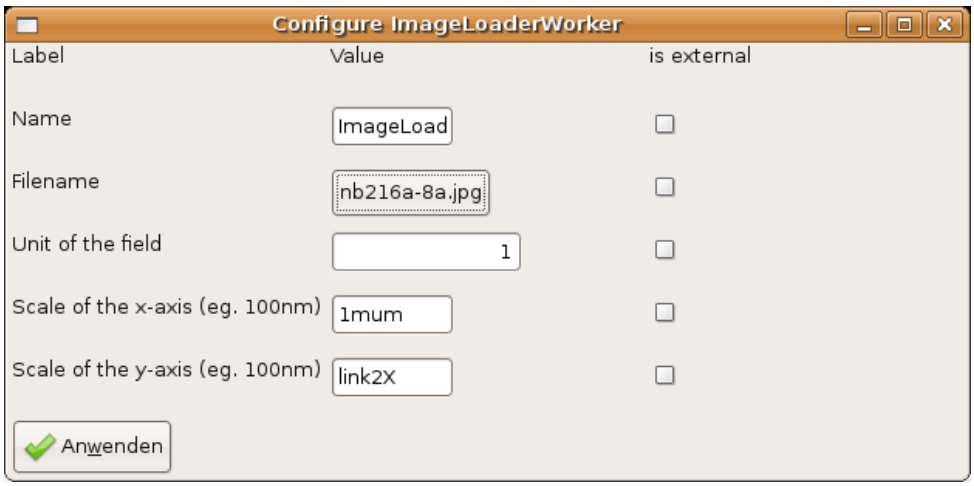


Figure 4. Configuration dialog for the ImageLoaderWorker which is automatically constructed from the definition of the ImageLoader class. It enables the interactive adjustment of all parameters, including the physical dimensions and units of the investigated sample. Note that the is external check-boxes are an experimental feature which allows the respective parameter to be set via a socket.

Please note the context menu emerging from the plug of the HistogramWorker. It enables the interactive examination of the computed results via visualisation plugins. Let's have a short look at the algorithm:

1. Loading the image

   Pyphant provides an ImageLoaderWorker which simply loads an image file from the location given in the workers configuration. The respective dialog can be opened by right-clicking the worker (Fig. 4). This scheme holds for all configuration dialogs of all workers. The loaded image is provided as a gray-scale image at the red plug. As the worker internally uses the Python Imaging Library (PIL[20]), it supports a wide variety of file formats.

2. Removing noise

   Next we want to remove noise from the image. For this task, the PILMedianWorker is applied, which implements a standard median filter[21]. It can be configured by the size of the applied kernel and the number of smoothing runs. Here we have chosen a 5x5 kernel and five smoothing runs.

3. Applying a threshold

   Now we want to separate the dark features which represent the PDMS phase from the background. This is achieved through the ThresholdingWorker. It compares every pixel of the smoothed image with a given threshold and returns a binary image such that the pixels which comprise features are set to 0x00 while the pixels of the background are set to 0xFF. In this example the threshold is set to 90. The threshold is chosen such that the fraction of the image being covered by features corresponds to the volume fraction of PDMS of the sample.

4. Measuring the size of the features

20. Secret Labs AB. *Python Imaging Library (PIL)*. http://www.pythonware.com/products/pil
21 cf. [1], p. 152ff

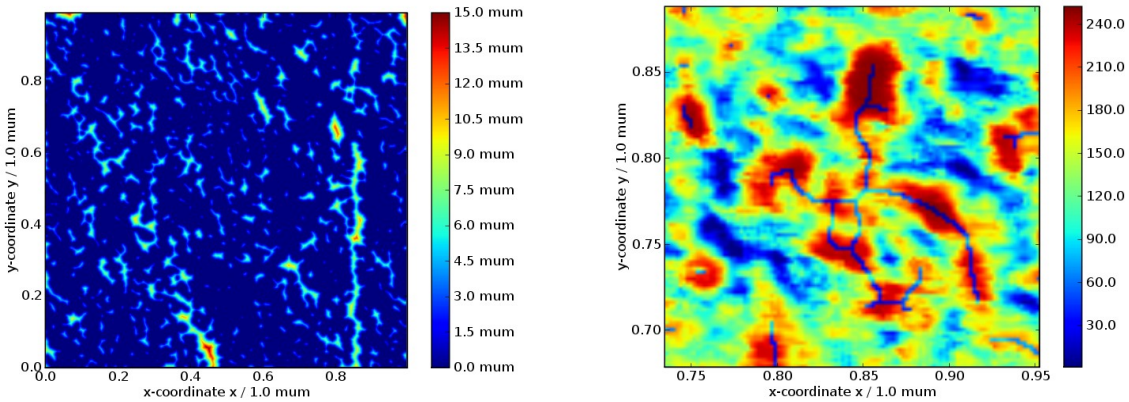


Figure 5. (a) Visualisation of DistanceMapper result. The feature size is colour-coded. (b) Display detail of difference between AFM image and skeleton of found features. Here the PDMS phase shows red while the skeleton is represented by blue lines.

By now we have a binary image representing the features we are taking into account. Next we would like to determine their size by calculating the distance of each pixel to the nearest background pixel[22]. This task is done by the DistanceMapper. The resulting gray-scale image is shown in Fig. 5a with pseudo-colours. Note the correct labelling of the colour palette indicating the distance of each feature pixel from the background.

5. Morphological transform

   In order to retrieve the width of the features, they are skeletonised. This is achieved by iteratively removing the outer pixels of each feature until the core pixels remain[23].

6. Checking result of skeleton computation

   The skeleton of the features can be compared with the primary data by feeding both images to the DiffWorker. A display detail of the result is depicted in Fig. 5b.

7. Determining the width of the features

   The skeleton of the features is applied as a mask to the distance map. This results in a skeleton image, where the brightness of each skeleton pixel corresponds to the width of the feature at the respective position. While this image is provided by the red plug, the blue plug returns the result as an Nx3 table representation. Here, each skeleton pixel is specified by its spatial position and the respective feature width.

8. Computing the histogram

22 cf. [1], p. 427ff.
23. Steven W. Smith. *The Scientist and Engineer's Guide to Digital Signal Processing*. California Technical Publishing, San Diego, 1997, Chapter 25. http://www.dspguide.com/pdfbook.htm

By now, the recipe produces the information we are interested in. The only thing left to do is compute a histogram from the data provided. This is done by the HistogramWorker. The resulting histogram presenting the length scale of the PDMS phase is shown in Fig. 6. The width distribution of the PDMS phase determined by the

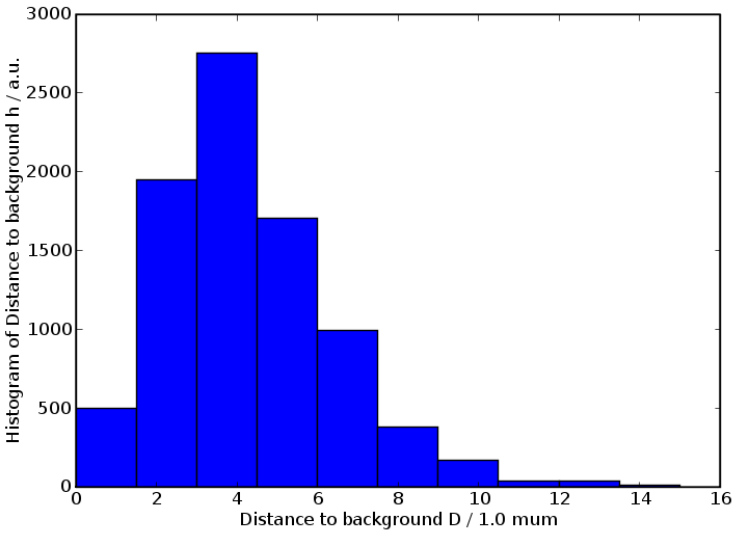


Figure 6. Width distribution of the PDMS phase. The result obtained from the AFM phase mode image depicted in Fig. 2 with the Pyphant recipe shown in Fig. 3. PILMedianWorker: 5x5 kernel, 5 runs. ThresholdWorker: threshold 90.

Pyphant recipe matches the results of Bruns et al[24].

4 The Pyphant Core

In this section we will describe the Pyphant core in greater detail. First we will show an example of a worker, then the worker base class in general. Next we will describe the connection facilities that link the workers into the recipe and the efficient computing model this suggests. Finally, we discuss the DataContainer, which is the preferred data exchange class. It is designed to maximise worker interoperability.

4.1 The Worker

4.1.1 The DiffWorker – a Practical Example

Listing 1 shows the DiffWorker. It takes two images and provides their difference image. First, some general attributes are declared (Lst 1, 1ine.44).

API is a hint to Pyphant on how to invoke the worker.

VERSION is related to the semantics of the worker. It should be increased whenever it is altered.

REVISION identifies the specific revision of the worker in the subversion repository.

name is the name of the worker as presented to the user. This would be the place for internationalisation (i18n). Omission of this information indicates an abstract worker, i.e. one that will not be presented to the user. An example for this is the fundamental Worker class.

Next, the sockets are declared. These are the input facilities of the worker. They are declared

24 cf. [19]

with a name and a type (Lst 1, line 48). The name is used to identify the socket. The type is used to provide visual hints to the user. Note that there is no declaration of output facilities. Those are referred to as plugs and are immediately coupled with a calculation method. From the implementors point of view, the leading parameters are ordinary parameters. If the worker is part of a recipe, Pyphant will call the method at a good time with appropriate arguments. The last parameter, subscriber, is special because it acts as a simple feedback facility for the progress meter. Therefore, the method should write information about its completion in percent at meaningful points in the calculation. Pyphant will supply a property-like object at runtime that is used to inform the user about the overall progress of the calculation.

Note that the names of the parameters coincide with the aforementioned socket names. This is necessary for Pyphant to figure out which socket should be associate with which parameter. The author of the worker is otherwise not required to deal specially with the input. The author simply declares a method and Pyphant takes care of all the data-handling necessary. All that is required is to prefix the plug with the Worker.plug decorator, declaring the return type (Lst 1, 1ine 51).

```
40 from pyphant.core import (Worker , Connectors)
41 import copy, scipy
42
43 class DiffWorker (Worker.Worker) :
44     API = 2
45     VERSION = 1
46     REVISION = "$Revision:␣28␣$"[11:-1]
47     name=" Diff␣Worker "
48     _sockets = [ ("image1" , Connectors.TYPE_IMAGE) ,
49         ("image2" , Connectors.TYPE_IMAGE) ]
50
51     @Worker.plug(Connectors.TYPE_IMAGE)
52     def diffImages(self, image1, image2, subscriber=0) :
53         im1=image1.data
54         im2=image2.data
55         diff=scipy.absolute(im1-im2)
56         result=copy.deepcopy(image1)
57         result.data=diff
58         result.seal()
```

Listing 1: The DiffWorker class which is included in Pyphant's ImageProcessing toolbox

4.1.2 The Worker Module

In section 4.1.1 we have presented an example for a simple worker, which like all of Pyphant's *productive* workers, inherits from the Worker class of the eponymous module (Lst 1, line 43). A small but important function, which the Worker module provides, is the plug decorator (Lst 2). It is merely used as a marker which adds the attributes `isPlug` and `returnType` to the plug-deploying method of the respective worker (e.g. Lst 1, line 51). This meta-information is used by the WorkerFactory metaclass (Lst 3) for gathering all plugs in the list `_plugs`.

```
51 def plug(returnType) :
52     def setPlug(plug) :
```

```
53              setattr(plug, 'isPlug', True)
54              setattr(plug, 'returnType', returnType)
55              return plug
56 return setPlug
```

Listing 2: Plug decorator of Worker module

Next, we have the worker class itself. Of special interest is the construction. At runtime, the `Worker.__init__` method will, for every worker instance, set the attributes: `_sockets`, `_plugs` and `_params`. All of them are lists, describing the respective requested objects. You have seen the `_sockets` list being filled in Listing 1. The list of plugs is constructed by the WorkerFactory metaclass on basis of the plug decorators (Lst 3, line 69). The `_params` list contains parameter descriptions and is filled like the socket list if the worker has parameters. Actually, parameters are a special type of socket, but we will come back to this in Section 4.2. For every entry in those lists `Worker.__init__` creates a corresponding connector instance as a member of the Worker instance.

```
64 class WorkerFactory(type):
65       workerRegistry=WorkerRegistry.WorkerRegistry.getInstance()
66       log=logging.getLogger("WorkerFactory")
67       def __init__(cls, name, bases, cdict):
68            cls._plugs=[]
69            for f in filter(lambda key : identifyPlugs(key, cdict), cdict):
70                 cls._plugs.append ((f,cdict[ f ]))
71            super(WorkerFactory, cls).__init__(name, bases, cdict)
72            try:
73                 WorkerFactory.workerRegistry.registerWorker(WorkerInfo (cls.name, cls))
74            except (AttributeError):
75                 WorkerFactory.log.warning("Ignoring␣worker␣"+name+"␣due␣
                       to␣missing␣name␣attribute.")
```

Listing 3: WorkerFactory metaclass of Worker module.

4.2 The Connectors

The Connectors module defines the type constants and the FullSocketError which is raised when someone attempts to insert a plug into an already used socket. Apart from that, the connector classes are also found here (i.e. Connector, Socket and Plug).

```
54 class Connector(object):
55      def __init__(self, worker, name, type=DEFAULT_DATA_TYPE) :
56            self.worker=worker
57            self.na me=name
58           self.type=type
59            self._isExternal=True
60      def _getIsExternal(self):
61            return self._isExternal
62      def _setIsExternal(self, isExternal):
63            if isExternal != self._isExternal:
64                  self._isExternal=isExternal
65                  self.worker.connectorsExternalizationStateChanged(self)
66      isExternal=property(_getIsExternal, _setIsExternal)
```

Listing 4: Connector class of Connectors module.

In Listing 4 you see the Connector class. It is the base class for sockets and plugs. As you can see, every connector carries a reference to its worker (Lst 4, line 56) as well as an identifying name (line 57). Furthermore, there is the `_isExternal` property (line 59) which denotes whether the connector is exposed to input from outside the worker.

The Socket takes the connecting plugs and keeps track of the connection. If any connection is broken, or the respective plug becomes invalid, the socket will invalidate itself and its worker, which in turn invalidates all of its plugs. This way, the invalidation propagates through the recipe until all concerned workers are informed.

Finally, we have the Plug. It is perhaps the most interesting connector, since it is the one responsible for multithreading. In order to accomplish threading, the framework needs a little help from:

1. the `Computer` class (Listing 5); and
2. the `createWrapper` method (Listing 6).

While the `Computer` class encapsulates the thread running the various calculation tasks, the `createWrapper` method is used at the construction time of the plug to create a matching wrapper for the calculation method of the worker. To this end, it constructs a method that starts one thread for every socket used by the plug and joins them back with the main thread prior to calling the plug itself, with the fetched results as its arguments. When a plug is queried via its `getResult` method, it checks for an already available result, generates a new one if necessary and handles the required locking transparently.

```
68 class Computer(threading.Thread):
69     def __init__(self, method, **kwargs):
70         threading.Thread.__init__(self)
71         self.method=method
72         self.kwargs=kwargs
73         self.result=None
74     def run(self):
75         if self.method:
76             self.result=self.method(subscriber=self.kwargs["subscriber"])
```

Listing 5: Computer class of connectors module.

```
88 def createWrapper(self, method):
89     args, varargs, varkw, defaults=inspect.getargspec(method)
90     sockets=args[:-1]
91     name=method.func_name+'PyphantWrapper'
92     l='def '+name+'(subscriber, method=method, process=self):\n'
93     for s in sockets:
94         l+='\t'+s+'=Computer(method.im_self.getSocket("'+s+'").getResult,
           subscriber=subscriber)\n'
95     for s in sockets:
96         l+='\t'+s+'.start()\n'
97     for s in sockets:
98         l+='\t'+s+'.join()\n'
99     l+='\tdef updater(percentage):\n'
```

```
100     l+='\t\tsubscriber.updateProcess(process,␣percentage)\n '
101     l+='\treturn method(␣subscriber=property(fset=updater),' # If
          no sockets are needed that comma will be erased, so do
          not add a space!
102     for s in sockets:
103         l+=s+'='+s+'.result,'
104         l=l[:-1]+')\n'
105     exec l
106     return eval(name)
```

Listing 6: createWrapper helper of connectors module.

4.3 The Pyphant Execution Model

How is a Pyphant recipe executed? Actually, it is not so much executed as evaluated. The naïve approach to execution might be to determine an execution order for the graph, then execute each node in a top-to-bottom order. Pyphant instead starts from the bottom node and fetches the required results of previous calculations. For example, the `UltimatePointsCalculator` provides two results:

1. An image that shows the found extrema visually; and
2. a list of the found extrema

While in a pure computer-oriented recipe the list might be needed for further processing, it can be convenient to have immediate visual feedback on the success of the operation, e.g., to determine the usefulness of the image preprocessing. However, only the requested result is calculated, thus saving time and computing power by avoiding the calculation of the entire node. Furthermore, this order of execution allows for an easy caching of already computed results: when a plug is queried it simply provides the last computed result without even bothering the worker, unless it has been invalidated meanwhile.

Another feature of this execution approach is the simple implementation of multi-threading. In case a plug has no or only an invalidated result, Pyphant retrieves the data from every socket used by that plug in parallel, each in its own thread. Thus a non-trivial recipe automatically leads to a pseudo-parallelised execution within the restrictions imposed on Pyphant by the *global interpreter lock*[25].

4.4 Information Exchange and Visualisation

In most cases, the scientific community deals with normalised physical quantities in the form of pure numbers. This is of course very convenient because it allows the immediate application of a wide variety of methods and algorithms. The disadvantage is that the scientific information is stripped of its meaning and is reduced to pure data, such that the labels of a result-presenting graph have to be recompiled by hand if the primary data or the applied algorithm change.

Therefore we were seeking a self-descriptive data exchange format which encourages the annotation of data right from the start and enables the entrainment of the physical units involved in the algorithm, such that a well-labeled graph can be produced without effort. The result is the DataContainer module which reproduces the self-descriptiveness of the network Common Data Form (netCDF[26]) but is augmented in the following respects: once sealed, a

25. Peyton McCollough. *Basic threading in python*. http://www.devshed.com/c/a/Python/Basic-Threading-in-Python, 2005.
26. Unidata. *netcdf (network common data form)*. http://www.unidata.ucar.edu/software/netcdf, 2003

DataContainer is immutable and can be identified by its **emd5** attribute (Enhanced MD5). This is a unique identifier composed of information about the origin of the container, its type and its MD5 hash[27] to ensure its integrity. It is also used to store the container as part of a recipe in an hdf5 file. Note that Pyphant does not impose any restrictions on the data exchange format, such that the described DataContainer is *just* a convenient interface for exchanging scientific information between workers (e.g., Lst 1) and visualisers (e.g., Lst 7).

The module DataContainer provides the basic class DataContainer which has the following attributes:

`longname:` Notation of the data, e.g. 'electric field', which is used for the automatic annotation of charts.

`shortname:` Symbol of the physical variable in LaTeX notation, e.g. `E_\alpha`, which is also used for the automatic annotation of charts.

`id:` Identifier in Enhanced MD5 (emd5) format
`emd5://NODE/USER/DATETIME/MD5-HASH.TYPESTRING`
which is set by calling the method `seal` and indicates that the stored information is unchangeable.

`label:` Typical axis description, composed from the meta information of the DataContainer.

`data:` Data object, e.g. Numpy array[28].

On top of the DataContainer we have defined a `FieldContainer` and a `SampleContainer` class. The FieldContainer stores an *n*-dimensional array together with its unit and the coordinates of the independent variables, which are called dimensions and in turn are represented as FieldContainers. The SampleContainer combines different FieldContainers which have the same number of sample points to a table-like representation.

The FieldContainer is characterised by the following properties:

`data:` Numpy.array representing the sampled field.

`unit:` PhysicalQuantity object[29] denoting the unit of the sampled field.

`dimensions:` List of FieldContainer instances describing the dimensions of the sampled field.

`data:` Sampled field stored as Numpy.array.

`error:` Absolute error of the sampled field stored as Numpy.array.

A SampleContainer can be regarded as a table which is typically obtained from measurement campaigns. It stores different observations on the same subject per row, whereby each column comprises a quantity of the same kind. From a statistical point of view, each row is the realisation of a random variable (sample). A SampleContainer is constructed from a list of FieldContainers and provides the following attributes:

`data:` table of samples stored in a Numpy.recarray;

`desc:` description of Numpy.dtype of the recarray; and

`units:` list of PhysicalQuantities objects denoting the units of the columns.

An example of a visualiser using the meta information provided by the FieldContainer is given in the following listing. A graph compiled by this visualiser is depicted in Fig. 5. From line 40

27. R. Rivest. *The md5 message-digest algorithm*. http://tools.ietf.org/html/rfc1321, 1992
28. Travis E. Oliphant. *Guide to Numpy*. Trelgol Publishing, http://www.tramy.us, 2005
29. Konrad Hinsen. *Scientific python.* http://dirac.cnrs-orleans.fr/plone/software/scientificpython/, 2007

of the ImageVisualiser module, you can see that we utilise the pylab module[30] in order to visulise our results. However, in order to make a visualisation class available to Pyphant's GUI, it has to register itself via the `DataVisReg` registry (Lst 7, line 74), from which the context menus of the plugs are constructed by means of the respective `name` attributes (Lst 7, line 45).

While the graph itself is created by a simple call to pylab's `imshow` procedure, the majority of code is spent on annotating the graph: (Lst 7, line 57):

`11.52-55:` Determining the reference coordinate system from the dimension attribute of the FieldContainer

`11.58-59:` Labelling the axis with the meta information of the dimension attribute

`11.61-70:` Creating the colour bar, which maps the false colours of the image to the amplitude of the visualised field.

```
40 import pylab, scipy
41 from pyphant.core.Connectors import TYPE_IMAGE
42 from pyphant.wxgui2.DataVisReg import DataVisReg
43
44 class ImageVisualizer(object):
45       name='Image␣Visualizer'
46       def  __init__(self, fieldContainer):
47           self.fieldContainer=fieldContainer
48           self.execute()
49
50       def execute(self):
51           self.figure=pylab.figure()
52           xmin=scipy.amin(self.fieldContainer.dimensions[0].data)
53           xmax=scipy.amax(self.fieldContainer.dimensions[0].data)
54           ymin=scipy.amin(self.fieldContainer.dimensions[1].data)
55           ymax=scipy.amax(self.fieldContainer.dimensions[1].data)
56
57           pylab.imshow(self.fieldContainer.data, extent=(xmin , xmax,ymin, ymax))
58           pylab.xlabel(self.fieldContainer.dimensions[0].label)
59           pylab.ylabel(self.fieldContainer.dimensions[1].label)
60
61           class F(pylab.Formatter):
62                def __init__(self, container, args, **kwargs):
63                     self.container=container
64                def __call__(self, x, pos=None):
65                     try:
66                          return str(x*self.container.unit)
67                     except IndexError, error:
68                          return str(x)
69           ax=pylab.gca()
70           pylab.colorbar(format=F(self.fieldContainer))
71           pylab.ion()
72           pylab.show()
73
74 DataVisReg.getInstance().registerVisualizer(TYPE_IMAGE, ImageVisualizer)
```

Listing 7: Excerpt from the ImageVisualizer module showing the homonymous class, which is used for visualizing the results depicted in Fig. 5.

30. John Hunter. *Matplotlib*. http://matplotlib.sourceforge.net, 2006.

5 The User Interfaces

We employed a clean encapsulation of the core of Pyphant. This allows for a variety of user interfaces. For now, we have implemented a simple GUI based on the wxPython toolkit. You already got a glimpse at the GUI in Section 3. In this section we will elaborate on the technicalities of the GUI, which also gives a good example of a Pyphant application. Then we will discuss a short example on how to incorporate a Pyphant recipe into a simple Python script.

5.1 wxPyphant – the Graphical User Interface

A screenshot of the GUI is seen in Fig. 3. On the left hand side of the client area of the window you see the canvas. This is where you put the desired workers and link them. The available workers are discovered at startup of the Pyphant framework via the entry points of the respective Python eggs[31] and are presented at the right hand side of the wxPython window.

The arrangement of the workers and their connections are conducted via an intuitive drag-and-drop interface. By placing a worker onto the canvas, the following mechanism is triggered: the factory method provided by the corresponding WorkerInfo object is called in order to construct a worker of that kind. Then a corresponding GUI object is created and integrated into the recipe. Sockets are represented by coloured boxes at the upper-left corner of the workers, while plugs are represented by coloured circles in the lower-right corner. Plugs and sockets of the same colour can be connected by left-clicking the plug and releasing the mouse button over the socket. Apart from the construction of recipes, wxPyphant allows for the immediate inspection of intermediate results by right-clicking the appropriate plugs. Upon a right-click, a context menu is shown that offers all suitable visualisations. Finally, the GUI features the saving and loading of recipes to and from hdf5 files.

While the GUI as a whole is based on the wxPython toolkit, the canvas is based on the Object Graphics Library (OGL), which in its latest form is part of wxPython.

5.2 Scripting with Pyphant Recipes

Thanks to the encapsulation of the core, Pyphant does not depend on the availability of a graphical environment at all, which allows the user to deploy recipes visually crafted on workstations into a more powerful computing environment. This is especially important for more complex tasks where the computation can easily take days.

Concerning the recipe discussed in Sec 3, the following Python script demonstrates the simplicity of re-using a Pyphant recipe with modified parameters. The only modules to be imported are: the PyTablesPersister (Lst 8, line 1) for loading the recipe; and ImageVisualiser (Lst 8, line 2) for saving the result of the analysis as a Portable Network Graphics (PNG) image. Once the recipe has been loaded, the individual workers can be accessed by the `getWorkers` method of the recipe (Lst 8, line 8). A Worker can be configured by setting the `value` attribute of the respective parameters (Lst 8, line 10). In order to obtain the result from a specific worker, the `getResult` method of the respective plug has to be called (Lst 8, line 14). The resulting DataContainer can be used to initialise a suitable visualiser instance (Lst 8, line 17) whose diagram can be exported to a suitable graphic file format (Lst 8, line 18).

```
1 import pyphant.core.PyTablesPersister
2 from pyphant.visualizers.ImageVisualizer import ImageVisualizer
3
4 #Load recipe from hdffile
5 recipe = pyphant.core.PyTablesPersister.loadRecipeFromHDF5File('demo. h5')
```

31 The PEAK Developers' Center: PythonEggs, http://peak.telecommunity.com/DevCenter/PythonEggs, 2007

```
 6
 7 #Configure ImageLoaderWorker
 8 inputWorker = recipe.getWorkers('Image␣Loader')[0]
 9 imageName = 'demo.png'
10 inputWorker.getParam('filename').value=imageName
11
12 #Fetch Result
13 worker = recipe.getWorkers('Apply_Mask')[0]
14 result = worker.plugCreateMaskedImage.getResult()
15
16 #Visualise Result
17 visualizer = ImageVisualizer(result)
18 visualizer.figure.savefig('result−'+imageName)
```

Listing 8: Simple Python script interfacing the Pyphant recipe depicted in Fig. 3.

6 Summary and Outlook

Pyphant is a flexible Python framework for the composition of data-flow models. It offers easy integration of new computing nodes and a multithreading execution of entire workflows without special burdens on the user. Its scriptability allows for the application of carefully crafted recipes in a computing environment under the lack of graphical services or possibly the integration into completely different applications. The current stable version of Pyphant is 0.4-alpha4 which is the basis of this paper. The framework and the worker toolboxes are published under the BSD license on Sourceforge[32] and in the Cheese Shop[33].

Concerning the application of the Pyphant framework, we are planning to extend the `ImageProcessing` toolbox with more tools and provide a toolbox for solving ill-posed problems on the basis of non-linear regularisation methods[34]. Possibly the family of toolboxes can be extended even further to entirely different projects in need of a similar GUI. To circumvent the restrictions imposed by the global interpreter lock and to harness the full potential of a parallel processing environment, we plan to refactor Pyphant into a compute server architecture, which is already hinted for by the present architecture. Furthermore, we are going to incorporate the emd5 identifier of the DataContainer into a *processing history* such that the origin of each result can be backtracked to the actual realisation of the underlying algorithm and the processed original data.

Acknowledgement

The authors would like to thank the editors for the opportunity to publish an updated version of our paper presented at the Europython 2006 conference[35] in *The Python Papers* and would like to encourage everybody who finds the presented framework interesting to participate in the project. The authors would also like to thank Michael C. Röttger for creating Pyphant's logo, Nico Bruns and Josef Honerkamp for fruitful discussions on the topic. The financial support by the German BMBF (Project No.: 03C0354A) is gratefully acknowledged. Finally, Andreas W. Liehr likes to thank Yanara M. L. Kempa for napping in a baby carrier at his chest while the introduction was written.

32. Klaus Zimmermann and Andreas W. Liehr. http://sourceforge.net/projects/pyphant/, 2007.
33. Python Software Foundation. *Python cheese shop*. http://cheeseshop.python.org/pypi/, 1990-2007.
34 cf. [7]
35. Klaus Zimmermann, Lorenz Quack, and Andreas W. Liehr. *Pyphant – a python framework for modelling reusable data processing tasks*. Refereed Paper Track, CERN 2006, http://tinyurl.com/r4rdz, 2006.

Upcoming Events

The following events, taken from the python.org events wiki[36], are being held between August and December this year.

September 7, 2007: Leipzig, Germany, Python-Workshop "Python im deutschsprachigen Raum 2007"

September 8-9, 2007: Birmingham, United Kingdom, PyCon UK

September 12, 2007: Cologne, Germany, monthly pyCologne meeting

September 15-16, 2007: Houston, Texas, Texas Regional Unconference

September 22-23, 2007: Leipzig, Germany, Python course for programmers (in English) taught by Python Academy.

October 8-10, 2007: San Francisco, CA, (Comprehensive) Introduction to Python course taught by Wesley Chun

October 10, 2007: Cologne, Germany, monthly pyCologne meeting

October 13, 2007: San Francisco, CA, Internet Programming with Python seminar+lab taught by Wesley Chun

October 20-21, 2007: Leipzig, Germany, Python course for programmers (in German) taught by Python Academy.

October 23-25, 2007: Longmont, Colorado, Python training class taught by Mark Lutz.

November 5-9, 2007: Atlanta, Georgia, Python Bootcamp taught by Dave Beazley at the Big Nerd Ranch

November 14, 2007: Cologne, Germany, monthly pyCologne meeting

December 12, 2007: Cologne, Germany, monthly pyCologne meeting

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.

36 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 clarify the 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.