PET English Translation

The magazine of the Argentina Python User Group

Nº 2
Nov 2010

Special Edition PyConAr 2010

ISSN 1853-2071

http://revista.python.org.ar

Issue 2: June 2011

In This Issue

Staff

Editors: Roberto Alsina, Emiliano Dalla Verde Marcozzi, Juan Bautista Cabral, Tomas Zulberti, Mariano Guerra, Diego Sarmiento, Manuel Arguelles, Juan Fisanotti, Marcelo Martinovic, Elías

Site: http://revista.python.org.ar

PET is a magazine created by PyAr, the Python Users Group of Argentina. To learn more about PyAr, you can visit our website: http://python.org.ar

What is PyConAr?

Authors: Tomas Zulberti & Juan B Cabral

I was editing the articles for this special version of the magazine and I realized something: except to those that attended, no one knows what is PyConAr, so this article intends to give you an idea about it.

PyConAr is an event in which several talks are given on topics related to the Python programming language and its name comes from a homonymic conference that is held regularly in the US named just “PyCon”. Despite the US event being a few years old, our first PyConAr took place in September 2009 in Ciudad Autónoma de Buenos Aires.

Now… what was PyConAr 2010?

Last year, the conference congregated us in Córdoba, at the Universidad Empresarial Siglo XXI. It involved 4 kinds of events:

- Talks: 20 minutes of dissertation followed by 5 minutes for questions. During the event 3 talks took place at the same time in 3 different rooms; and between each group of talks there were 15-minutes breaks if one wanted to talk to the speaker. There were a total of 43 talks in PyConAr 2010.
- Keynotes: the name is self explanatory (in English), all assistants were gathered in a big auditorium to witness important guests’ expositions. There were 3 of these talks in the event (1 on Friday and 2 on Saturday) by the end of each day.
- Lightning talks: organized during the same day of the conference. The two biggest differences to normal talks were:
    - 5 minutes long, no time for questions
    - They didn’t have to be related to Python. You could talk about anything.
- Posters: This was the first PyConAr that had posters. They were the summary of a project in form of a poster (sic) exposed in a area with a high volume of assistants going by so they could be read during breaks.

But the conference was not just about Python. It was also about people; it is an opportunity to see each other’s faces, those people that reply to you in the mailing lists or in IRC chats.

PyConAr was an event also about work, fun, comradery, friendship and, who knows, maybe even some couples here and there. But that is PyAr, a group of crazy people that love what they do and create things while having fun. Thanks to everyone for making PyConAr a reality and thanks to PyAr for existing and giving a lot of people a place to share their passion.

Links

- http://nqnwebs.com/blog/article/pyconar-2010-el-orgullo-de
- http://fisadev.blogspot.com/2010/10/pyconar-2010.html
- http://blog.elrincondemariano.com.ar/2010/10/pyconar-2010-fue-todo-un-exito/

How PyConAr2010 was made

Author: Horacio Francisco Sebastián Del Sagrado Corazón de Jesus Duran Barrionuevo,

aka perrito666, with anonymous contributions in the PyAr Wiki page.

Hitchhiker's guide to make a PyConAr2010

The PyCon 2010 was gone as it came, they all seem very happy at the end. These almost 10 months of work left me thinking how terribly easier would have been things for this event if I had known the things that I know now. So, towards the next person that has to do this difficult but noble task, here comes "The PyCon tutorial"

First off a tip, if you do not value a unique moment in the community and the fact of creating a nurturing space for people to share more than things like money and comfort, you might want to think twice before start with this event. Sure you will lose a lot of time and probably some money and you will sleep just a little in last month, having almost nothing to give to your stomach health and leisure time.

The tutorial only has the organizational and the little details of the event, technical part we will go out together during these months and writing down everything in the meanwhile, much of the software to generate things there needs to be rewritten, I'll also help with that.

Team selection

Before thinking about merchandising, locals, thematics and other things, you will have to think about a team, the cornerstone of each event. Beyond the people necessary to conduct the event in the day, you'll need a team that is willing to go with you anywhere and put the same amount of grip for several months.

In my experience, you need:

- 2 people for merchandising and printing of graphics.
- 1 person for the design (assuming you have a base with which to work indicating that type of graphics is needed and the data it carries).
- 1 system administrator (if you use a system, in our case PyConAr, need someone to handle every time it explodes)
- 1 webmaster (someone who upload content on the site, you will need to provide a lot of information and do not have time to look for it and process yourself)
- 1 community manager and press officer (sounds dumb for an event, but it takes a lot of time to spread through social networking and other)
- 1 treasurer / accountant / juggler (these things move a lot of money and at the end of the day you want clear accounts)
- 1 in charge of sponsors (it is confusing for companies that you do not have 1 univocal person to contact, a sponsoring must be handled with the person who made the first contact)
- 1 responsible for negotiating the coming of the keynotes.
- 1 talks reviewers coordinator.

A few tips to keep in mind:

- A friend can be a light, but not necessarily a good partner: When you have a partner, you have to be able to delegate a task and expect it to take place in a timely manner.
    - Delegation is not to ignore:
        - You have to keep in mind that if a problem arises you have to be ready to lend a hand to your partner to get back on track.
        - You have to pass all the necessary data to your collaborators so they can do their work and make intelligent decisions.
        - If you delegate a task or decision that requires resources beyond the role of the person, you have to be available to answer his/her concerns.
- Enthusiasm is no skill or time: Although when you decide to make a event of considerable magnitude and we have many very interesting prospective collaborators, not all are candidates to coordinate tasks.
    - Many have more enthusiasm than time and do not understand the consequences of breaching any of its assigned tasks, so one usually has to be the person that faces consequences.
    - Sometimes the arrangement and responsibility do not come with the ability to make the necessary decisions in time to act. In this case delegating can backfire, since we will end up spending more time answering questions and taking decisions than if we had done the task initially.
- Youth and inexperience is not synonymous with immaturity and inability: Young people in your group are often more in touch with current reality and you have more free time than those already working full time and have families or

occupations.

After all choosing people is not so difficult, only keep in mind the perceptions of your local community and find out about past experiences of them, the foundation of our community is the reuse of past experience.

If you still have doubts after all this you can try with some small tasks to ensure that your impressions are correct.

Location

It is not simple to choose a place, especially when we do not know how much people are going to stay. We try to choose a place where maximum people that could come to the event will fit, watching events from the past with similar features helps. Measuring against events in Capital Federal is not a good idea, unless you do an event in Capital Federal, they always bring more people together. If some version numbers are available about events in interior it is better to use those to calculate. Another very important thing to keep in mind is how difficult / costly is to have internet connection there. You will need to think about:

- Ability to build wiring: Distribute internet for a couple of hundred people is hard enough, you usually need to lay cables APs carry everywhere, Please keep in mind when you negotiate the place if it is possible to use pre-existing wiring or put a new one. To see what you do need USLA has a “box of wonders” of about 70kg of equipment for network infrastructure, electricity and other necessities for events.
- Possibility of an ISP: The number of people your are going to have is rarely controllable in the network (beyond the control your router device can do) so you’ll need a good internet connection. Make sure that an ISP can provide this connectivity in the event, the first option is to try to get the ISP of the event, if any, enlarge the capacity of the service for a few days. Try to not use a residential service, they tend not to endure.
- Ability to load the Electrical Infrastructure: most likely you will go to plug several power strips with notebooks. Make sure that the main electrical panel of the site will not blow up when you connect a lots of notebooks, projectors, audio amplification equipment, APs, switches, etc.. If you can bring your own power protective devices.
- Accessibility: Please note that software communities tend to be very different, find out if the building, at least in the parts you use, is ready to receive people with motor disabilities without external aid. Also make sure that area is accessibility-friendly, nothing serves a wheelchair lift that serves to upstairs in a temple at the top of the Himalayas.
- Empowerment: Verify whether any special empowerment is needed for the event, a municipal process, etc.
- Security: If we re-arrange other events after that we’re planning, make sure you can do it free, doing it behind bars usually brings drawbacks … Read: we are responsible of the event (and its possible consequences to others).
    - Electrical Safety: Many building’s installations are not ready for ”individual users” plugging their devices in the electrical network, they have industrial installations with other security requirements. Make sure that public outlets have differential protection. The responsible for the room must know where and how to cut the power in case of accident. It would be a good practice to include in the contract of the place that has adequate protection (to determine responsibilities)
    - Injury Prevention: Special care must be taken especially when there are stairs, there should be no wires and objects that may cause tripping as loose wires, etc.. .
    - Evacuation Plan: In many places it is mandatory to have stuck in places visible traffic direction to evacuate the site as well be identify the emergency exits. Volunteer firefighters can help us in this. Once chosen the place and confirmed, make sure that what you’ve been promised being written and recorded in all means they deem necessary, is important in this case meeting the needs of the bureaucracy. Registration of agreements and respect for the place protocols help to survive changes authorities and other contingencies.

Sponsoring

Assuming that you chose a place, you got a date and safe equipment that will help, it’s time to go get money. It is the custom of free software communities in Argentina that the events are no charge. This is good point of view of the low barriers to entry, but it causes many problems on other fronts, I try to tell my experience and one from PyConBr to help you face them.

- An event without tickets is not a commitment on the part of many potential participants: prepare for enough numbers of registered exceed the number of attendees. The problem is that one usually has to plan several factors, which often involve investing money, depending on the number of attendees. In a paid event the no assistance does not involve such a problem, since the payment was made and this covers at least part of the cost incurred by that person. Try the inscription to be detailed and that it is clear to people that their registration is not only an act of support, triggering a series of actions on your part. Also trying to reach the audience regularly to confirm attendance.
    - ”’Tip:”’ PloneConf2010 people created a mailing list that included all participants, it is interesting to share expectations about the event, including logistics and organization and send general announcements and get them discussed, besides the good feedback it provides.

- A free entry event does not project serious plans to businesses: This may cost you some sponsors, particularly those who are more “old school “to see an event as a place of recruiting and training. On the side of recruiting they don’t see that your barrier to entry is as high enough to be useful. On the side of training, they don’t believe that with so little cost your course will be high profile.
- Anyone can access: While we ask for a fairly complex form to enter, we always end up allowing access at the entrance. Due to the volume of people trying to credit we do not ask many details for the registered in situ. If possible do this in a place where you can restrict access to most people only having an identification of the event. When you get to someone who is not registered, form a queue until you have credited pre-registereds. Take all the necessary data from people not registered, you got a commitment to providing the safest place / comfortable as possible for the people can develop their community business as freely as possible. If you got the human capacity put a specific person to register unexpected people and someone checking the entry for strangers.
- If possible implement a system of mixed entrance, this should please both parties: companies and just community. For people who wants to register as assistants of a business you could implement a fee, a change in the badge could be different for attendees sent by companies, with the name of it. You can add more things to do stuff more attractive as a customized badge if they send more than N persons or number of tickets if they buy a sponsoring. Remember always that facilitate people coming is as important as buying a sponsorship, people make the events interesting. If you get a company also send its developers you got the auspices implicit and you got something to show others. As for the people in the community or that they are individuals (whether or not from a company) admission is clearly free. I have not implemented this personally, but it’s worth exploring and extending the idea.

The second major issue about Sponsoring is selling sponsoring plans. After a long conversation with Dornelles Tremea and Érico Andrei on finding sponsors, some items are worth sharing.

Regarding the level of costs, there is two types of company:

- The FLOSS-related SMEs or technology for your event: They will put the chest to the bullets and are in favor of what you’re doing and give you a stronger hand because they share your ideology rather than interest in advertising (which also have). Often can not get rid of a high amount of money, but are interested in helping you out. Its good to have a special pricing to them, you can arrange exchanges also with them, perhaps you need to have something besides money. Always remember to rinse the reason for this pricing plan, to understand that implies a certain commitment to the cause other than the sponsoring purely commercial.
- Large Business: For this company, although it’s hard to believe, the more expensive the more attractive. There is an important component of circus in these things, they want to show they are up to the situation and head of mainstream events, their way of measuring these is:
    - Turnout.
    - Cost (the sponsoring and entrances, as discussed earlier)

This brings us to third important issue of sponsorship: For many of the sponsors your event is not the same as for you, they often do not share the ideological baggage. The event is a product, has more to do for them advertising, image, business recruiting and other things. Like any product, to sell, you need a seller. Get a person with sales skills, if possible, professional. This is one of the points which I think is more than valid, even critical, to invest money. You could fix a percentage of the proceeds in exchange for services. The person to deal with the sponsors should understand the codes that they handle, to read and anticipate the needs that will be attractive for each one of the potential investors. Might you think that the community can do everything, but I promise not. To deal with a commercial area or PR do not send a technician, these people do not buy a product, they purchase the seller. After checking the actual experience of people of PyConBr (they outsource almost throughout the event after many years) I understand that an experienced salesperson can give much more scope and visibility. The more approach and sponsors you have, the better the quality of the event you can arrange for community, think about passages of interesting speakers who can not afford travel, scholarships, information material, food, coffee, all these things are good in one event and cost money.

Finally a piece of advice: Try to match your prices to similar events elsewhere, in case of PyConAr, would be good that prices and performance of the sponsors, while it’s not disruptive to the spirit of the event, are balanced (except the obvious economic differences both countries).

Information

This was probably our weakest point during the 2010 event. We should not underestimate the importance of information for people that comes from abroad and need to calculate costs, time and logistics in general. Ideally by the time you confirm a place and going to make a official call, you will need to provide, in the most clear and complete possible, the following data:

- Site Address: The address of the venue, accompanied by a map, if possible a mapping system of the type of google maps or openstreetmap, so that people can play with it and become familiar with the place.

    - Landmarks: References to the site from different landmarks of the city (Monuments, shopping centers, universities, etc.), many may have an idea of some of these places and this helps them settle better.
    - Urban Public Transport: All urban transport that passes moderately close, the name of the stop and how to get from there to the exact point of the event. Always is good to clarify which of these mediums go from airports, bus terminals, ports, etc. where might be more interested in getting visitors from other cities. Do not forget the price and payment method of urban transport.
    - Interurban: How to get at least from the most important cities in the country to yours, would be good to name at least one transport for each province. There is no such thing as too detailed, the more information the better. It is very important for all these transport you also include the price.
    - International: How to get to your city from other countries, if you include some kind of international arrival make it clear. The nearest town that can be reached by plane or other international transport (It goes that also put something on how to get from this city to yours, a link to interurban at least)
    - Parking: Many people asked if they could park close in the last event, try to relieve the nearest parking and put some information on where they are along with their price and plans (hourly wages, weekly, etc.)
- Accommodation: Many people come from outside and prefer the word of a local when choosing accommodation. As you are in home probably you won't not know much about accommodation, after all you live there and there are not many reasons to go to a hotel often;).
    - List of hotels by type: It's good to have a list of hotels and all contact information, arranged by number of stars and type (hotel, hostel, Little House on the Prairie, etc.). Including the price is not bad idea. When you make the relieve do not forget to ask if they place some kind of plan groups, many are very cool with this. Give them your contact details and website address where you will put your data, hotels tend to call to inform if they change their prices or availability, this saves work at the time of updating information, take into account that you will spend a few months between the relieve and the event.
    - Media for local lodgers: One of the advantages of these events is get to know each other and share, try to create a space for those coming from outside and people with room to accommodate get in touch.
    - How to get from the hotels: Add some of the transport information page of the hotels, at least basic and further away.
- Food: Add to all this information a variety of places to eat near the event, please keep in mind the special needs such as celiac or vegetarian. Some information about notable places in the town to eat is not bad, can serve to those that come earlier and want to know a little about the town. Again I remind you, prices.
- Contents: Try to keep the information of the talks and spaces available. Don't let pages with errors on your site, instead put something that says it is not available and when it will, try to meet that date, if you are not certain about the date, do not promise impossible.
- Dates: Make sure the registration deadline, calls for talks, surveys and other matters that requires third-party interaction have visible deadlines and repeat as necessary.

Accounting

Money is always an uncomfortable subject, especially when the event is a matter of community and friendship or companionship.

Economic predictability

For this area is important to know some facts:

- Develop at least three quotes with different levels of optimism, you'll to help prioritize your expenses and allocations of money.
- Set deadlines for confirmations and sponsors payment thereof. It is important to maintain the economic uncertainty the furthest time as possible to the event.
- Prioritize in the budget the things you've promised to the sponsors, is a commitment to the community that supports you and our reputation is important to our existence.

Finances

Whichever method you have to manage money officially (we had help from a foundation that gave us your account and legal to deliver receipts) you have to take very clear accounts.

Some tips:

- Before you begin make sure that all who will handle money, who sells the sponsorings and you know very well what are the details of billing for expenses to carry out the details of receipts that are who will give money and qualifications of the case. Some core are:
    - What type of receipt is given to the sponsor and what information from it will you need.

        - What kind of bill(s) you can accept and what data you have to deliver for this (I recommend you do a paper with this information and take it in wallet from the first day until you close the event)
        - How long does it take to issue receipts.
        - How payments are made to the event from the abroad.
- Make sure the person handling the money have all the information needed to decide where the money goes
- Make a weekly meeting to decide how to allocate budgets and know how things are going.
- Maintain a shared spreadsheet with data for:
    - Sponsors who promised to buy packages (with information on when each pays)
    - Cost of each item to purchase (with information about when the payment was made and if you have the invoice)
    - Movements of money from third-party destined to the fund of the event (sometimes one has to make an upfront payment to the event)
    - Totals:
        - How much money you were promised
        - How much you got physically
        - How much money you owe
        - How much did you say you would spend
        - The spent.
- The invoice of everything you spend is NOT optional and no money is given if no invoice. If any official organization has to account for their movements will need the appropriate invoice for each expenditure. You got to be intransigent with this or it can cause a lot of problems. Demand that the bill be as detailed as possible, it is your right and you will simplify your life.
- Make a financial check prior to each session and during the event end of each day. You need to know that you did not do any expenses that you forgot to write down (happens a lot with petty cash when you are very close to the event)

Selection of speakers

There are two major types of speakers:

- Keynotes: Invited by the organization to speak, this is the icing on the dessert of the talks, the show part of the event. Some tips to take into account.
    - Anticipation: The people you invite is often important in community and has a tight schedule. Try to have the names ready and make the invitation as early as possible (about 10 months is fine.)
    - Popular choice: Make a list of three times over what you need and let people from the community sort by interest. Needless to say when you create the list you may see that for bringing these people, for example, that you do not have to pay a Luis Miguel combo to everyone to come. It’s good that people can choose their heroes :) and also give you options for backup persons in case the main guest could not come.
    - Guest Profile: Deserved importance to this is not always given, but it is very important. I think in general you enjoy when the international guest enjoy. If you bring someone who is going to be a rockstar throughout the event it can cause an uncomfortable situation for everyone:
        - Check his/her community involvement and communication with others, usually easy to see if is someone social (for geek social standards)
        - The language, if you bring someone from a country that speaks another language you can look if you can communicate enough. If you speak his/her language is better so the guest can enjoy the event. Try that guest talks about something that at least a handful of your colleagues speak, or the situation can be awkward.
        - Special Needs: This is something to be reckoned with everybody, but as this person is your responsibility during his/her stay, make sure you meet special health needs, food, beliefs, etc.. (Eg do not bring a person with respiratory problems to a very moist city or a veggie to eat an “asado”)
- General lecturers: if you can make a survey on issues that matter most for the election of talks, this sure will help (or make the selection prioritize). If a talk is very interesting and aligned with what appears to be the interest of the participants consider afford to travel the lecturer if it can not.

Pre-event logistics

As you approach the event, the workload increases, some of the following tips help you get in a timely manner and without an ulcer.

Merchandising of the event: Once you have decided which will be the merchandising just take a week to find out the following from it:

- Costs: (I guess you choose the best value for money)

- Processing time: You will need this to know when you have to order everything.
- Place of delivery: This is important to know what kind of logistics transport you have to manage.
- Packaging: You have to know where are you going to store this

With all this information and assuming that your financial calendar allows it (It should if you started to organize the event well in advance) seek order items twice the processing time before the event (Assuming the item is something that lasts in time and that time is not ridiculously long). This formula does not always doing so as a rule, you can not have anything later than 1 week before the event.

If possible, start putting together packages when you got the items, this saves you from last-minute runs. If someone offers to take the extra material demand same times as you demand you to yourself.

IMPORTANT NOTE: If you make souvenirs for visitors or speakers, make sure it is something that can pass the customs and can be carried in an aircraft.

Artwork: Get a printing house for things that you are going to need and when you get it make sure you they are comfortable working with several formats files and are open to help and suggest materials, processes and other things. Trying to stay with the same people always speed up procedures. If you have to print something like a book (in our case the python tutorial) get a printed version from who was the layout designer of the book (as well you get something to give examples) and an experienced printing house, printing this kind of thing is not trivial.

Badges and custom material: Try to get personalized material be the minimum, if badges for example, there is always a part that can be done the same for all, using stickers or something similar to complete during the event. If anyway you can have this one week earlier is better.

Guests, lecturers and keynotes: There are some things to foresee to make life easier to guests:

- Send to all these a signed letter in their language and the most detailed and official as possible as sometimes are required at customs and at work.
- Book a hotel to who are your responsibility, make sure to send a mail with details about what includes the stay.

Certificates: Make all the necessary certificates and sign them prior to the event, it’s easier to destroy undelivered certificates rather than sign 200 in a day.

Original version

Original version of this article can be found at http://python.org.ar/pyar/HGTTP

Lightning Talks at PyConAr 2010

Author: Juanjo Conti

A Lightning Talk is a short presentation chosen by the speaker, no more than five minutes long.

Unlike the Scheduled Talks, there is no approval process; the speakers just enroll during the PyConAr. The topic of the talk doesn't have to be related to Python necessarily and the speaker can talk about anything.

This rapid period guides the speakers to focus on the essentials and provides the audience about ten topics for one hour.

The good thing about the lightning talks is that you can get to know cool things in just 5 minutes. And if you are listening to a ho-hum speaker, it doesn't matter: he/she will leave in 4 minutes!

Talks

During the PyCon we had lightning talks all the time, Friday and Saturday. These were some of them, along with links to follow up:

- Diego Sarmentero about NINJA IDE: http://www.ninja-ide.org.ar
- Manuel Naranjo about cusepy: http://code.google.com/p/cusepy/
- Manuel Naranjo about OpenProximity: http://www.openproximity.org/
- Mariano Guerra about embedded Python: https://github.com/marianoguerra/talks/raw/master/PyConAr2010/lightning.pdf
- Federico Heinz about a campaign against electronic voting: http://trac.usla.org.ar/e-votrucho
- Roberto Alsina about a transformative Web browser.
- Roberto Alsina: "A spreadsheet in N lines".
- Juanjo Conti showing up a demo about Taint Mode: http://svn.juanjoconti.com.ar/dyntaint/trunk/webdemo/
- Alejandro Cura about Deferreds.
- Martín Gaitán about version 3 of PyAr t-shirts: http://python.org.ar/pyar/RemerasV3/
- Luciano Bello about Graphological documentation indexing: http://people.debian.org/~luciano/security-doc/
- Naty Bidart about tests.
- Ricardo Kirkner about django-configglue: https://launchpad.net/django-configglue
- Roberto Allende with How to cook the Wikipedia using two eggs.
- Emiliano Della Verde Marcozzi about SQLAlchemy.
- Hugo Ruscitti showing up a component-oriented framework for gaming: http://www.pilas-engine.com.ar/doku.php
- Nueces about PyCamp.
- Felipe Lerena about Mozilla Argentina: http://www.mozilla-ar.org/
- Tenuki y manuq about Karma, a scoring system for PyAr list: http://listas.python.org.ar/listinfo/Karma
- Joaq about PyAr Wiki.

Keynotes Pictures

Author: Juanjo Conti with pictures from the Machinalis's album

Keynote

What is a keynote? In this kind of event a keynote or plenary talk is the moment where everything is halted. Even though during the day there are talks in different rooms, when keynotes happen everybody is met in the same room, for example in an auditorium to listen to a well-known speaker. In general, the speaker is some international guest of relevance in the topic being treated.

At PyConAr 2010 the keynotes speakers were Leah Culver about the Python's Zen applied to business ventures and Christiano Anderson about MongoDB.

Leah

Culver is an engineer in software specialized in Web applications development and the Django framework.

She was co-founder and leader of the social network Pownce, which was acquired by Six Apart in November 2008 and the website was shutdown.

She is passionate about open source and open specifications, in particular OAuth and OEmbed. She likes to taste new programming languages and she is now developing an iphone application for Plancast.

She lives in San Francisco though she grew near Minneapolis.

Personal Web Site: http://blog.leahculver.com/

Christiano

Anderson is member of Python Brasil and developer of Free Software at Trianguli. He has been consultant for ten years, specialized in web technologies especially Python and Django.

Has been speaker in several events in Brazil and the abroad.

His presentation is available at: http://www.slideshare.net/canderson/python-and-MongoDB

Personal Web Site: http://christiano.me/

More event pictures

Pictures in this article are from http://www.flickr.com/photos/machinalis/sets/72157625148609499/

PyConAr2010’s barbecue

Author: Juanjo Conti, with words stolen from Fisanotti & Batista.

This is how it was announced on September 8th:

On Friday, October 15th at 8pm there will be a meeting around a barbecue to which all PyConAr 2010 speakers and assistants are invited.

In the barbecue, the intention was to reveal the social side of the event, a place to play ball, relax and chat.

The barbecue, or meal in the end, is normally after the last day of talks, but in this occasion, with Mother’s day and return trips in the middle, a lot of people would be leaving Saturday afternoon so the moment to (in no particular order) socialize, think back, enjoy and share happened in the middle of the event.

The barbecue is an excuse

How’s that a barbecue is not a barbecue? The original idea was to have a barbecue, but it ended up being something that appears to be top-of-the-art in Córdoba: “pata” (leg). The term describes a service provided by some people that arrive with a huuuge cow leg, already cooked, lots of bread, some containers with different sauces and they cut the leg “live” right there and then so that you can make yourself little sandwiches to your taste. It was not clear to me if the lettuce and tomatoes was provided by the organizers or were part of the “pata” service. —facundobatista

Through a new member’s eyes

It is great to see so many people engaged with the advocacy and evolution of a technology, to the point of donating work hours, long travels, giving their knowledge away and even economic contributions to the cause; convinced that it is something that is completely worth it. Coming from not-so-distant past immersed in other technologies (Microsoft, .Net, etc.) it still amazes me what can be accomplished by a community with no hierarchies, official titles nor companies supporting it.

And there's also the human aspect, the one-on-one interaction. What you share during the most informal moments, like the barbecue, or going out on Saturday is priceless. Conversations and discussions with people that, besides being professionals, are real people. Those that have no problem in sharing the solution to an issue, knowledge, and advice as well as jokes, good spirit and a beer here and there ;) (though I didn't drink, I still find it cool LOL). —fisadev

Pictures are courtesy of:
http://www.flickr.com/photos/54757453@N00/sets/72157625061739525/

Understanding Decorators in Python

Author: Juanjo Conti
Juanjo is a Systems Engineer. He has been programming in Python for 5 years now and he uses the language to work, investigate and have fun.
Blog: http://juanjoconti.com.ar
Email: jjconti@gmail.com
Twitter: @jjconti

A note about what's next

Next article is not an article properly speaking, it is the *relief* of a talk that was presented in oral form, not from an audio medium but from my memory! I hope the reader will find it useful.

Introduction

The topics I treated in the talk are the following ones:

- The beginning of everything
- What is a decorator?
- Decorator functions
- Decorators with parameters
- Decorator classes
- Decorating classes

The beginning of everything

Everything in Python is an object

- Identity
- Type
- Value

In Python everything is an object. Integers, strings, lists, tuples and other weirder things: modules are objects, the source code is an object. Everything is an object. EVERYTHING.

Each object has 3 features or attributes: identity, type and value.

Objects

Let's see some examples. Number '1' is an object. Using the built-in `id` we can find its identity. Its type is `int` and its value is obviously 1.

```
>>> a = 1
>>> id(a)
145217376
>>> a.__add__(2)
3
```

Because it's an object, we can apply to it some of its methods. `__add__` is the method that is called when we use the `+` symbol.

Other objects:

```
[1, 2, 3]   # lists
5.2         # floats
"hello"     # strings
```

Functions

If everything is an object, so are the functions.

```
def greeting():
    print "hello"
```

We can obtain a function's id using `id`, access its attributes or even make another name refer to the same function object:

```
>>> id(greeting)
3068236156L
>>> greeting.__name__
'greeting'
>>> say_hello = greeting
>>> say_hello()
hello
```

Decorator (non-strict definition)

Let's take a liberty for a moment and say that a decorator is a *function* **d** that receives as parameter another *function* **a** and returns a new *function* **r**.

- d: function decorator
- a: function to decorate
- r: function decorated

We can apply the decorator using a functional notation:

```
a = d(a)
```

Let's see how to implement a generic decorator:

Code

```
def d(a):
    def r(*args, **kwargs):
        # a's previous execution behavior
        a(*args, **kwargs)
        # a's after execution behavior
    return r
```

We define a function **d**, our decorator, and in its body is defined a new function **r**, that we are going to return. In the body of **r** **a** is going to be executed, the decorated function.

Now change the comments for code that actually does something:

Code

```
def d(a):
    def r(*args, **kwargs):
        print "Start of execution of", a.__name__
        a(*args, **kwargs)
        print "End of execution of", a.__name__
    return r
```

When executing a decorated function with the above decorator, a bit of text will be shown, then the decorated function will be executed and more text will appear as it finishes.

In `sum2` we keep the decorated version of `sum`. Now see what happens when we execute it:

Manipulating functions

```
def sum(a, b):
    print a + b
```

```
>>> sum(1,2)
3
>>> sum2 = d(sum)
>>> sum2(1,2)
Start of execution of sum
3
Stop of execution of sum
>>> sum = d(sum)
>>> sum(1, 2)
Start of execution of sum
3
Stop of execution of sum
```

Also we can keep directly in `sum` the decorated version of `sum` and now the original version will not be longer accessible.

Previous way of applying a decorator is the *functional* form. We have a nicer one:

Syntactic sugar

As of Python 2.4 the @ notation was incorporated for function decorators.

```
def sum(a, b):
    return a + b

sum = d(sum)
```

```
@d
def sum(a, b):
    return a + b
```

In the above code snippet you can see two examples where we compare the different ways of applying a decorator.

Next there are some examples of *real* decorators.

Warning

Counter-example: the evil decorator.

```
def evil(f):
    return False
```

```
>>> @evil
... def something():
...     return 42
...
>>> something
False
>>> something()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'bool' object is not callable
```

This is a cheater decorator, because instead of returning a new function, it returns a boolean object. Obviously when we try to run it, we get an error.

Chained decorators

Their application is similar to the mathematical concept of function composition.

```
@register_use
@measure_execution_time
def my_function(some, arguments):
    # function's body
```

Is equivalent to:

```
def my_function(some, arguments):
    # function's body

my_function = register_use(measure_execution_time(my_function))
```

Decorators with parameters

- They allow more flexible decorators.
- Example: a decorator that forces a return type of a function.

Suppose we want a decorator that converts to string all a function's returns. You could use this way:

```
@to_string
def count():
    return 42
```

```
>>> count()
'42'
```

How would you implement that? See a first approach:

```
def to_string(f):
    def inner(*args, **kwargs):
        return str(f(*args, **kwargs))
    return inner
```

This way works, but think if we can do this in a more generic way. Below you'll find the use form of a decorator named `typer`:

```
@typer(str)
def c():
    return 42
```

```
@typer(int)
def age():
    return 25.5
```

```
>>> age()
25
```

In fact, `typer` isn't a decorator, is a decorator factory.

```
def typer(t):
    def _typer(f):
        def inner(*args, **kwargs):
            r = f(*args, **kwargs)
            return t(r)
        return inner
    return _typer
```

Note that `_typer` is the true decorator, the outside function gets a parameter `t` which is used to define the nature of the decorator to be created.

Now go further and see:

Decorator classes

Features:

- Decorators with status.
- Better organized code.

The first example is similar to our first decorator function:

```
class Decorator(object):

    def __init__(self, a):
        self.variable = None
        self.a = a

    def __call__(self, *args, **kwargs):
        # a's previous execution behavior
        self.a(*args, **kwargs)
        # a's after execution behavior
```

An example of how to use it would be:

```
@Decorator
def new_function(some, parameters):
    # function's body
```

Step by step operation:

- A `Decorator` type object is being instantiated with `new_function` as argument.
- When `new_function` is called the method `__call__` of the instantiated object is executed.

We can also apply it using the old notation:

```
def new_function(some, parameter):
    # function's body
new_function = Decorator(new_function)
```

Because of these examples you've seen, we can make a more strict definition of decorators:

Decorator (more strict definition)

A decorator is a *callable* **d** that gets as parameter an *object* **a** and returns a new object **r** (in general of the same type as the original or with the same interface).

- d: object of a type that defines the method `__call__`
- a: any object
- r: decorated object

```
a = d(a)
```

Decorating classes(Python >= 2.6)

As of Python 2.6, the @ notation is permitted before a class definition. This gives place to the concept of class decorators. Even though if before 2.6 you were able to decorate a class (using functional notation), recently with the introduction of this syntactic sugar the class decorators audience got bigger.

A first example:

Identity:

```
def identity(C):
    return C
```

Returns the same class we are decorating.

```
>>> @identity
... class A(object):
...     pass
...
>>> A()
<__main__.A object at 0xb7d0db2c>
```

Change a class totally:

```
def abuse(C):
    return "hello"
```

```
>>> @abuse
... class A(object):
...     pass
...
>>> A()
Traceback (most recent call last):
  File "", line 1, in
TypeError: 'str' object is not callable
>>> A
'hello'
```

Similar to one of the examples you've read at the beginning, this example shows us that a decorator's return has to have a similar interface as the object we are decorating, that way it makes more sense to change the use of the original object, for a changed one.

Replace with a new class:

```
def replace_with_X(C):
    class X():
        pass
    return X
```

```
>>> @replace_with_X
... class MyClass():
...     pass
...
>>> MyClass
<class __main__.X at 0xb78d7cbc>
```

In the previous case we see that the class was changed completely for a brand new different class.

Instance:

```
def instantiate(C):
    return C()
```

```
>>> @instantiate
... class MyClass():
...     pass
...
>>> MyClass
<__main__.MyClass instance at 0xb7d0db2c>
```

As last example of class decorators we've seen a decorator that once applied, instantiates the class and links this object to its name. This can be seen as a way to implement the Singleton design pattern, studied in programming. # wikipedia singleton quote

To finish:

Where can I find decorators?

Permissions in Django

```
@login_required
def my_view(request):
    ...
```

URL routing in Bottle

```
@route('/')
def index():
    return 'Hello World!'
```

Standard library

```
classmethod, staticmethod, property
```

Thank you very much!

The talk finished thanking the public for their attention. I take this opportunity to thank César Portela and Juan BC for reading this *relief* draft.

Data and contact

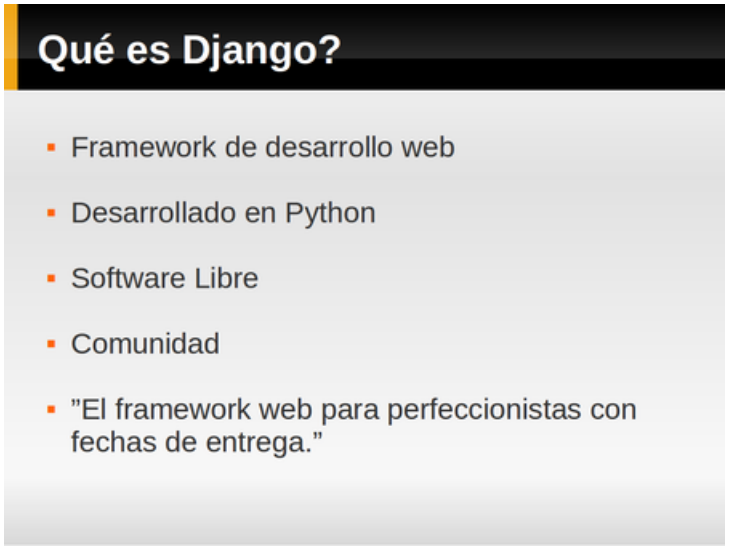
PyConAr 2010 - Córdoba - 15/10/2010

- Comments, doubts, suggestions: jjconti@gmail.com
- Blog: http://www.juanjoconti.com.ar
- Twitter: @jjconti
- http://www.juanjoconti.com.ar/categoria/aprendiendo-python/
- http://www.juanjoconti.com.ar/2008/07/11/decoradores-en-python-i/
- http://www.juanjoconti.com.ar/2009/07/16/decoradores-en-python-ii/
- http://www.juanjoconti.com.ar/2009/12/30/decoradores-en-python-iii/
- http://www.juanjoconti.com.ar/2010/08/07/functools-update_wrapper/
- Original slides: http://www.juanjoconti.com.ar/files/charlas/DecoradoresPyConAr2010.pdf

Introduction to Django

Author: Juan Pedro Fisanotti
Python is my favorite language, though I'm quite fond of Ruby and Lisp; Django is the web framework I like the most, Linux user (Ubuntu) from not too long ago, and free software enthusiast. Nowadays I mostly work with Python :)

The original talk was titled "Mini-introduction to Django" but the written version wasn't so "mini".



So, ¿what's Django? In a few words: it is a web development framework. That is to say: a set of libraries and tools that will allow us to create websites. As you probably imagine, it's written in Python so it will also be Python (with all its benefits) the language that we use to create our websites.

Django is free software, so we have access to its source code to learn, understand, enhance it, etc. It also has a big and active community that helps keep it up to date, have bugs detected and squashed, an updated and detailed documentation, and some other advantages we'll later see (spoiler: a lot of useful applications already developed!).

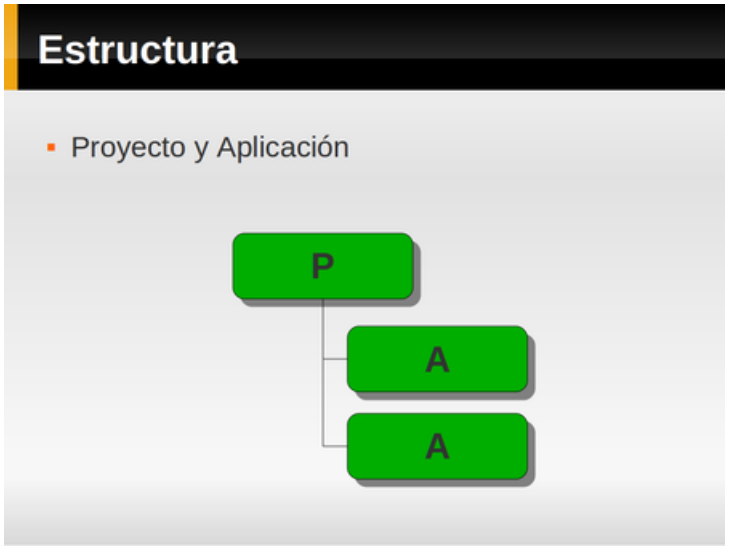
What's more, it is good to know that Django has a well-defined philosophy, influenced by the environment it came from. The original Django creators worked building sites for news companies, where most of the times changes were needed within hours or days. Being a "perfectionist" group of developers their challenge was meet deadlines writing code the right way and not compromising to get it out the door faster.

Its from there that comes the saying that Django is "the web framework for perfectionists with deadlines". Django not only aids us in developing, it also actually helps us write good code.

Finally, it is a framework that tries to be flexible, not get in between the developer and what he wants to accomplish. That's why it is very simple replace the parts of Django we don't like for others we like more or are more useful.

Introductions out of the way (provided I didn't bore you to death and caused you to stop reading), let's get to know Django.

The first important thing to know is what Django proposes for our sites' structure:



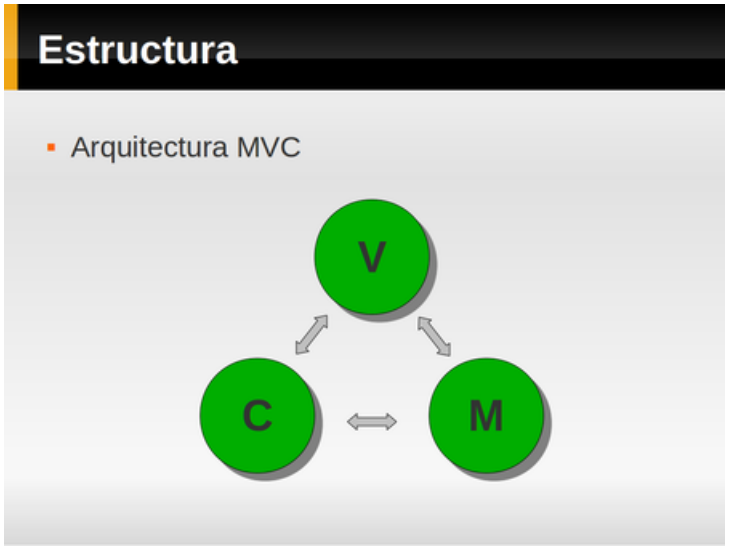
In Django we will have Projects and Applications:

- Project: it will contain our site's general configuration (things like the database, administrator's emails, etc.) and a set of applications.
- Applications: they will be the ones containing the functionality itself for our sites (for example, the logic to find the most voted puppy picture this week).

Something interesting to know is that Django promotes applications to be as independent as possible so that an application can be re-used in more than one project.

Django itself already brings a lot of useful applications for common web-development tasks like user authentication or content administration. And, having a community so huge (and humble as you can see), we can also find third-party applications to re-use and modify.

There are applications that will help during the development (logging, debugging, etc.), applications to add functionality to our website (tagging, registration, etc.) and pre-built applications to directly deploy in production (like blogs, CMS, etc.).



Now, what does an application look like?

For our applications Django proposes the Model-View-Controller (MVC) architecture. To those that have not heard about it beforehand, Django didn't invent MVC, it is a well-known architecture that proposes a 3-fold division of our applications:

- Models: the part of our applications that define the structure of the database and takes care of interacting with it.
- Views: the user interface with the code that chooses what data to ask for or show each time.
- Controllers: the part of the application that picks views to show as a response to the users actions or requests.

Models will be classes that represent things that we want to store in the database. Example classes: Client, News, etc.

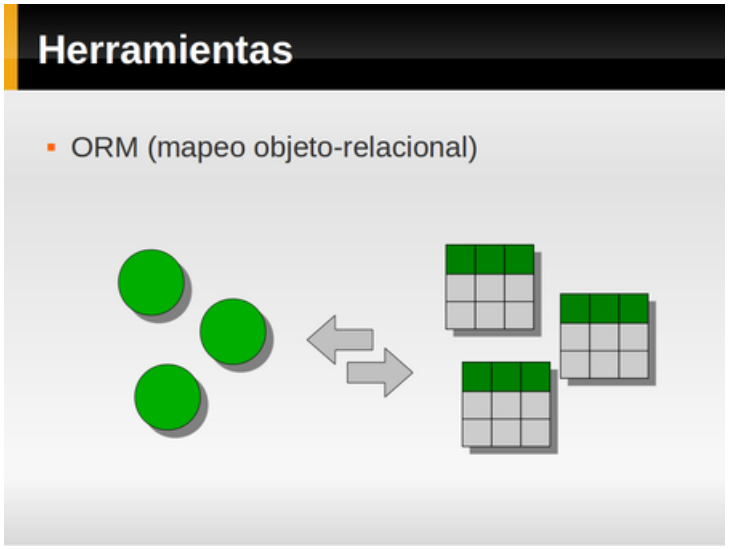
Views will be regular Python functions that will return the content that needs to be delivered to the user (webpage, image, etc). For example: the view "main_page".

And for controllers, we just need to define what function (view) is to be called for each URL. For example: "when the user asks for the URL http://misitio.com/start/ we run the view "main_page".

Django will take care of most of the controllers and provides us with tools to ease the development of models and views.

Lets take a look...

The Django package includes:



The first tool Django provides is an “Object Relational Mapper” (ORM). The ORM will allow us to interact with the database so it will be one of the two things that we will commonly use the most (so pay attention, this is like something that will be in the final for sure).

Like we are using object-oriented programming, we are going to define and use classes. Like most commonly used databases are relational databases, Django will take care of translating our operations on objects into SQL statements that will run on the databases’ tables.

For example: we define our class User with several properties (name, address, email, etc.). We can then do things like create instances of User, store values in its properties and tell it: “hey, you, save yourself!”, Django will automagically create the appropriate update or insert SQL statement and run it in the database.

The code for this example could look like this:

```
new_user = User()
new_user.name = 'Fisa'
new_user.email = 'fisadev@gmail.com'
new_user.address = 'somewhere in Argentina'
new_user.save()
```

Just like this, the ORM allows also us to read from the database, filter, run complex queries, etc.

For example, if we want to get all users that are 20 years old, it could be dome through some code like this:

```
past_teens = User.objects.filter(age = 20)
```

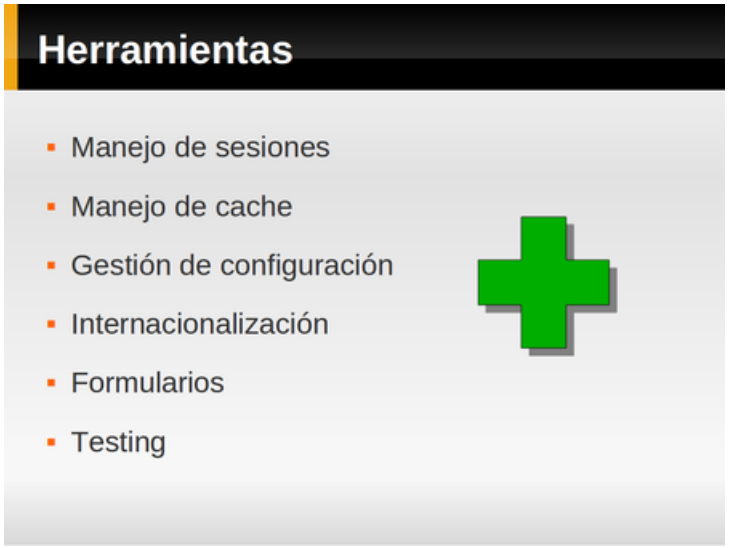
[while we are at it: Django’s built-in authentication application already includes a User class that we can take advantage of :)]

We can now write code that reads from and stores stuff into the database, but that is not of much use to the user. We have to show him something. The second toll we’re going to use is the templates system.

What’s that? It is what will let us “inject” our data into HTML (or XML, or whatever) templates that we define.

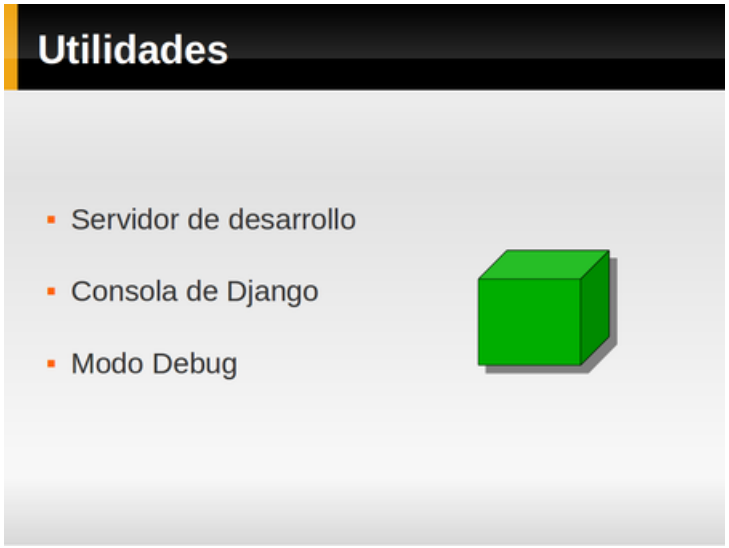
That is to say that we’ll build the HTML for the user to see in their browser but we’ll say within the file things like “here goes the name of the current user”.

The template language also provides some basic logic structures that we can use like loops, conditionals and even some kind of template inheritance.



There are also a number of extra tools that we will use intensely while developing with Django. It is always a good idea to get to know them so that we don't re-invent the wheel every time; they are also tools that have been thoroughly tested and polished.

If something smells popular, then it's probable that Django already has a tool to help with it. If there isn't one, it's very likely that someone developed it as an application. And if there is no such application, awesome! Now you found something you can contribute back :)

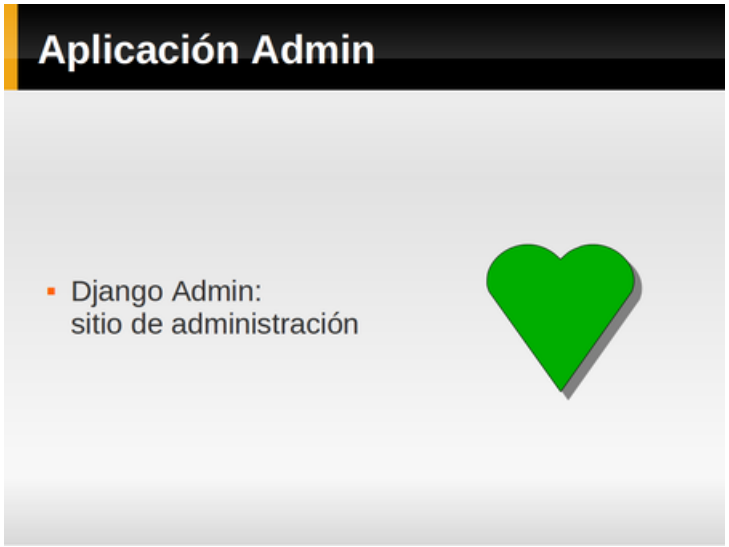


The tools we've seen so far are mostly things we will use only while we are developing. Things we will use within our code. But Django also comes with tools to help us with tasks beyond coding.

The development server will allow us to execute our website from the development environment itself, without the need to configure a web server for it or do complicated deployments for every change we make.

Django's console gives us the chance to run code in an Python interactive console, but just as if it was our site running in the server. We can use it, for example, to interact with the database with our models.

And there is the debug mode, that shows us a lot of information about the errors in our website's code execution. Having debug activated, when an error arises we will be able to see the code that generated it, the variables that were in memory at the time and their values, GET and POST parameters received, the site's configuration, Python's exception stack trace, etc. Those that have done previous web development know what that is worth as it is not normally available.



And to top off this list of tools and utilities, there is Django's Admin.

Django Admin is a builtin application that will save us a lot of work. I know it sounds exaggerated and fanatical, but my statement is quite realistic.

As you remember, we can define Models in our applications. Those are classes that represent the things that we store in the database (classes for User, News, Comment, etc.).

In the models we already defined everything needed to be able to operate with those entities in the database, create new ones, search, store, update, etc. So, what other thing

is needed to get an application that provides functionality to administer the database's content? Logically, screens (pages).

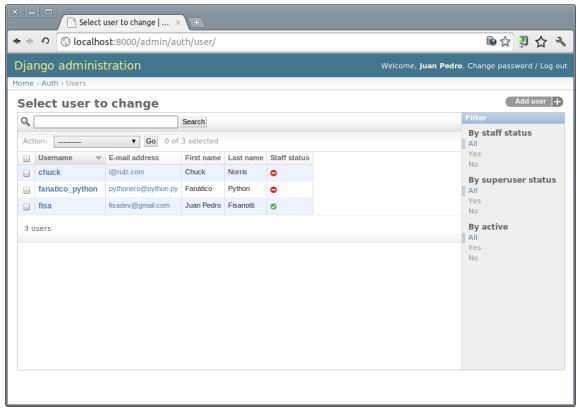
Well, Django's Admin is exactly that. We simply tell it something like "hey, admin! Look, here I told that I had customers in the database, can you give me a customer's CRUD?". And the Admin will do just that. It's that simple (if you don't believe me, check the code of Django's tutorial).

[CRUD is an application that allows you to Create, Read, Update and Delete]

We can even customize the way those forms are shown so that we can add filters on fields of our choice, decide what columns to show in lists, how to show fields when editing, etc.

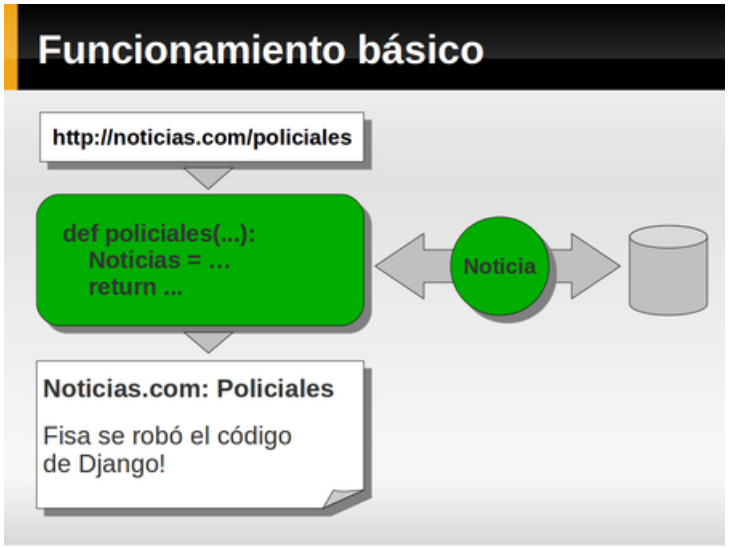
This application is of great use to us developers during the creation of the site to add content and get previews. It is also useful in production for the site's administrators or moderators. But it can also be of use to the end user, depending on the case.

Every time I see the admin, I regret the hours I devoted to developing CRUDs that are not a fraction as configurable as Django's :)



Making it all work

Well, we have now gone through a few interesting things Django includes. The question now is: how do we put all these things together?



It is easier than it looks:

1. The user asks for a URL ("I want to get http://news.com/crimes really fast!")

2. Django realizes that to serve that URL needs to call a function (a *view* from now on). As you imagine, it does call it.

3. In our view (the function "cops" for example), we get the news from our database that are categorized as related to cops.

4. Finally, our view will tell Django to return to the user a specific template (in this case, the HTML of the cops page) but using the data we provide within the template.

So far we have a general idea of how it works and how to work with Django, but it can be understood a lot better if we see it with code. It is for that reason that I recommend the official Django tutorial that you can find at http://docs.djangoproject.com/en/dev/intro/tutorial01/

In the tutorial, you can see how to build a simple application from scratch. With what you read in this article it will surely be even easier to understand what you will be doing there.

And if it is not easier, you have my mail to complain :)

PyAfipWs: facilitating, extending and releasing AFIP Web Services (Electronic Bill and others)

Author: Mariano Reingart
Programmer Analyst and Teacher. Enthusiast of Free Software and Python, PostgreSQL and in particular.
Blog: http://reingart.blogspot.com
Company: http://www.sistemasagiles.com.ar

Introduction

*PyAfipWs <http://www.pyafipws.com.ar/>*_ is a free software interface to AFIP Web Services, developed in Python.

It was born around year 2008 in some email exchanges in the mailing list of PyAr!, when with Mr Marcelo Alaniz, consulting about some problems with Python and webservices, we started to build a library adapted to the requirements and environments of AFIP (which by the way, are assorted and no tool in Python worked 100% on them).

The interface was “inspired” in the official examples of AFIP for PHP, which were the simplest ones to understand and closer to another language like Python. There were examples for Java but they were incomplete (some of them had difficulties with dependencies and incompatibilities with XML/SOAP) and in .NET (which were more complete but only for Windows). Anyway, for our taste, these two examples were pretty hard to follow, they used to generate code (“artifacts”), etc. etc.

Additionally, we inspired ourselves from PHP in some libraries of simple management of XML and SOAP, which led to its counterpart in Python because they seemed simple and intuitive.

Initially we developed it for the web services of authentication (electronic signature and others) and electronic bill, but with the passage of time we started to add other AFIP services such as capital goods - electronic tax bonus, export bills, grain traceability code, faithful depositary and the new services for domestic market.

The project has matured and some libraries we develop or adapt have been freed separately:

- PySimpleSoap (http://pysimplesoap.googlecode.com/): for simple yet complete management of webservices
- PyFPDF (http://pyfpdf.googlecode.com/): for PDF generation in an easier and fluid way.

Both have been integrated in web2py (http://www.web2py.com) to have a complete framework for developing management web applications.

Electronic bill in Python:

As an example, we will show how to authorize an electronic bill in few lines.

First and foremost an access ticket must be requested (using a certificate and private key previously requested), which will allow us to use the electronic bill web service:

```
from pyafip.ws import wsaa

# create access ticket, sign it and call the ws
tra = wsaa.create_tra()
cms = wsaa.sign_tra(tra, "homo.crt", "homo.key")
ta_xml = call_wsaa(cms, wsaa.WSAAURL)
```

Behind the scene M2Crypto is used (which is the binding of OpenSSL for encryption in Python), needed to sign the request to access in XML.

From the ticket we extract the Token and Sign (using SimpleXMLElement to analyze and convert the XML in Python objects and data structure), which contain the security codes required in other webservices:

```
# process xml
ta = SimpleXMLElement(ta_xml)
token = str(ta.credentials.token)
sign = str(ta.credentials.sign)
```

Once the authentication to access the web services is obtained, shall we proceed to request authorization to emit an electronic bill.

For that we create a connection to the server and call remotely the authorization procedure:

```
from pyafip.ws import wsfe

# create SOAP client
client = wsfe.SoapClient(wsfe.WSFEURL,
      action = wsfe.SOAP_ACTION,
      namespace = wsfe.SOAP_NS)

# authorize electronic bill (obtain CAE)
```

```
res = wsfe.aut(client, token, sign,
   CUIT=20234567393, tipo_cbte=1, punto_vta=1,
   cbt_desde=1, cbt_hasta=1,
   tipo_doc=80, nro_doc=23111111113,
   imp_total=121, imp_neto=100, impto_liq=21)

print res['cae'], res['motivo']
```

If everything went well, this will return the CAE (Electronic Authorization Code in Spanish), needed to make the electronic bill.

That would be all for webservices in Python. For generation of the bill (for example in PDF), email delivery and such, see PyRece below.

Legacy languages

Beyond the applications in Python, this library is compatible with languages such as Visual Basic, ASP, Fox Pro, Cobol, Delphi, Genexus, PowerBuilder, PHP, .Net, Java, ABAP (SAP), etc. and with any language/application capable of creating Component Object Model (http://en.wikipedia.org/wiki/Component_Object_Model) in Windows (for example Excel or Access).

This is easily accomplished using PythonCOM (part of the win32 extensions), wrapping a common class of python to be exposed to other applications, defining the methods and public attributes, the exposed name and other, for example:

```
class WSAA:
    "Interface to the WebService of Authentication and Authorization"
    _public_methods_ = ['CreateTRA', 'SignTRA', 'CallWSAA']
    _public_attrs_ = ['Token', 'Sign', 'Version', 'XmlResponse']
    _readonly_attrs_ = _public_attrs_
    _reg_progid_ = "WSAA"
    _reg_clsid_ = "{6268820C-8900-4AE9-8A2D-F0A1EBD4CAC5}"
```

Once the interface is registered, it can be called from any other application with this technology, for example, in Visual Basic it would be:

```
Set WSAA = CreateObject("WSAA")
tra = WSAA.CreateTRA()
cms = WSAA.SignTRA(tra, "homo.crt", "homo.key")
ta = WSAA.CallWSAA(cms, url)
```

```
Set WSFE = CreateObject("WSFE")
WSFE.Token = WSAA.Token  ' set token and sign of wsaa
WSFE.Sign = WSAA.Sign
WSFE.Cuit = "3000000000" ' issuer CUIT

ok = WSFE.Conectar(url)

cae = WSFE.Aut(id, presta_serv, tipo_doc, ...
           imp_tot_conc, imp_neto, impto_liq, ...)
```

In our case this was very useful and made possible to many applications contemplate these new features (webservices, encryption, etc.) with minor modifications, which otherwise would have been difficult or even impossible.

Text Files and Command Line

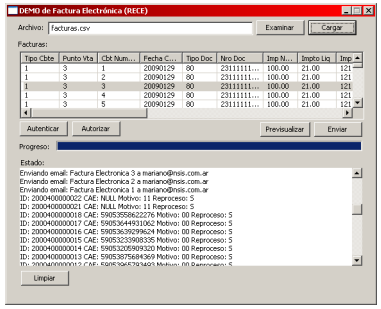
Even though the COM interface is very useful for relatively modern applications, there are still very difficult to access languages or environments, where practically the only way of interoperatibility are text files.

Languages as Cobol handle files with fields of fixed length (this was already supported by the RECE of SIAP app), we went down that road, which with Python was quite straightforward.

The interface includes tools like RECE.PY and RECEX.PY which receive and process input files by command line, saving results in output files. This behavior es controlled by a configuration file (RECE.INI) which defines URLs, certificates and paths to use.

PyRece: free SIAP?

The history doesn’t end here, some difficulties of the RECE Application or the online services to make electronic bills (limited functionality, delays or complexities, etc., especially for the contributors that do not posess management system) have been seen, we developed a visual application (“desktop”) to facilitate and extend the possibilities brought by AFIP:



User interface is a simple screen but it contemplates:

- CSV spreadsheet reading (instead of fixed width file)
- Real-time online Authentication and Authorisation
- Customised PDF generation (texts, logos, lines, etc.) with picture of the bill
- Email delivery to the customer of the PDF (with a brief configurable message)

All these features are not available (together) in the application or web site of AFIP (some of them can be done with limitations, but not in an agile and transparent way).

This application turns to show that PyRece has a concrete possibility of being developed in Python with a free Integrated Applications System (SIAP), in this case as alternative to the RECE application of electronic bill, but there are other cases where the same could have been done if web services were available (eg, IVA for DD.JJ., SICOSS for social security, etc.).

FE.py

Finally, we developed a tool that unifies the different webservices (national electronic bill, export, capital goods, etc.) and integrates all the aforementioned, using a database to store and exchange the information, generating the bills in PDF, enabling them to load from and to text files, deliver by email or FTP, amongst others.

In general it uses PostgreSQL, but other databases can be used (PyOBDC or SqLite).

The Project

Finalizing the article, we have included some comments about how the project was developed.

An already solved topic was the business model, especially knowing that there are other closed alternatives and we decided to keep ourselves in the path of free software, finding the right combination to be able to compete was not a minor issue.

For those who don't know Python and are eager to evaluate the interface, we provide an installer with a fully functional demo for testing environments. We offer paid commercial support for those who need to ask questions, request corrections and adjustments and want to have access to the installer for production environments and upcoming updates we might release.

All the source code is published in the repository of GoogleCode (http://pyafipws.googlecode.com) under GPL3 license, with all the needed scripts and installation instructions.

We think the result was quite satisfactory, allowing to extend and support financially the project, contributing to free software and the community with alternative and powerful tools, and beginning to create a developers group interested in the topic.

The interface has thousands of downloads from the project's web site, and many important companies hired us for commercial support.

This has not been done entirely without setbacks or effort, therefore to finalize there are some comments and recommendations:

- Installers and packages were critical to make people able to evaluate the products, especially for not widely spread technologies like Python, and mainly for Windows, which has been the largest market for this interface.
- Documentation and examples were another strong spot, and the experience says that is a constantly deepen topic, even in cases that it seems to be enough (we include frequently asked questions, code snippets, important remarks, etc.)
- Training curses and workshops are also very productive (we have had 2 in the ACP, to whom we are thankful), allowing to dig deeper on the topic and meet the interested people.
- For the dissemination we've been helped by blogs, news, free software events, etc. As the project appeared in the search engines, its popularity and real utility for the users grew.
- The community support in our case was not effective, mailing list and ticketing/issues systems were not heavily used, neither were there a lot of contributions or source code revisions, maybe for the sensitive and/or particular character of the topic (although the phisolophy of free software has not been widely spread in some environments).

We hope this article will be useful for a general overview of the topic, any additional information might you need can be found in the following addresses:

- Web site of the project: www.pyafipws.com.ar (http://www.pyafipws.com.ar)
- Newsgroup: groups.google.com.ar/group/pyafipws (http://groups.google.com.ar/group/pyafipws)

- Commercial support and Documentation: www.sistemasagiles.com.ar/trac/wiki/PyAfipWs (http://www.sistemasagiles.com.ar/trac/wiki/PyAfipWs)
- Minisite AFIP electronic bill: www.afip.gov.ar/fe (http://revista.python.org.ar/1/html/www.afip.gov.ar/fe)

InfoPython - Measuring the Value of Mass Media Information with Python.

Author: *Juan Bautista Cabral*
JBC met Python a lonely night of 2007. He developed his degree project of the Systems Engineering career with this language using the Django framework and worked 1 year developing information evaluators using our lovely reptile.
blog: http://jbcabral.wordpress.com
mail: jbc.develop@gmail.com

Infopython is a library for the evaluation of media information using formal theories from social sciences. Initially, they were a scattered set of modules used in my work, then in few days of refactoring and patience I managed to unify a single library.

Background

There are different sociological theories to determine the importance of media on public opinion, they analyze the information they emit from the viewpoint of sender, receiver, or both.

To quote one example, the **Information Theory of Shannon** is a mathematical formalization of a sociological theory known as the **Hypodermic Needle**.

In the case of **Infopython** theory under consideration is known as **Agenda-Setting** (in the future is planned to add others); which posits that mass media have great influence on the public when determining which stories have informative interest and how much space and importance is given to them. The focus of this theory is to assign a higher priority to obtain more **audience**, greater **impact** and a determined **awareness** about the news.

The Media

In **Infopython**, before mentioning **how** the processes are for calculating the value of information you need to determine **what** is what we are measuring.

That way we informally define that for our domain an information medium is:

One emitting element on which I want to make a measurement of its information value. This information has as characteristics: homogeneous, give a "feeling" of unity, and be measurable.

Being:

- **Homogeneous:** All information issued by the media must have common characteristics. But given its extreme internal environmental variation we find it impossible to measure.

  For Example:

  > a television channel for our case does not qualify as "medium" since each program and time slot has **very** different levels of audience and content; in this case is best to take as unit each TV show as the medium to measure.

- **Sense of unity:** It's easier to grasp this concept from an example:

  If I say that my information medium are "Magazines of Sport" it gives a feeling that this element is not "one medium", yet when I change the definition of the medium, to "Goals and Lanterns sports magazine" this feeling is present.

- **Measurable:** If from a medium that we defined we can not extract quantitative data, it makes no sense to us.

Formalizing

Reached the point where we have defined **what a medium is** for our domain, we can define "mathematically" a model that fits the previous definition of the "Agenda-Setting", and propose the following:

The value of information from one medium is a function of the audience and impact, discarding the conscience as it is difficult or impossible to measure

Or what is the same:

VALUE = F(AUDIENCE, IMPACT)

Being:

- **VALUE:** Is the importance of the medium given the theory.
- **AUDIENCE:** To how many people the information of the medium reach.
- **IMPACT:** How important is the medium to the audience.

Now is being proposed as **F**:

F(AUDIENCE, IMPACT) = AUDIENCE * IMPACT

The function **'*'** (Multiplication) is chosen due to:

- **Better reflects changing values:** if a parameter increases or decreases much, so varies the value.

  Let's assume the following case:

  A medium which is followed by a low number of audience **10** but has a high impact **1000**. This can occur if these few people belong to a group of influence

(presidential advisors for example).

In this case the value of the information would be **10,000**, and since we define value as “much” and what value we define as “little”, this value is “large” (much). This, we can consider as correct as any medium that can influence important people should have a high value.

Keeping the values, but changing our function by **AUDIENCE + IMPACT** the value of the information would be **1010** which, keeping the above reasoning remains high.

Now, if we replace the value of the audience by a large number **1000**, and keep the impact, the value of the original function would be **1 million** and the second case in **2000**.

If we consider we now likely impact on 1000 presidential advisors **2000** value remains small, since we are presence of a medium likely to generate a global impact. Which shows that multiplication represents much better variation of parameters.

- **The value vanishes in the absence of audience or impact**: When the audience or impact are **0 (zero)** (No one sees or pays attention to the medium) the value of information is also **0 (zero)**.

  This is not trivial because it suggests that the information is worthless if nobody cares to see it or no one pays attention.

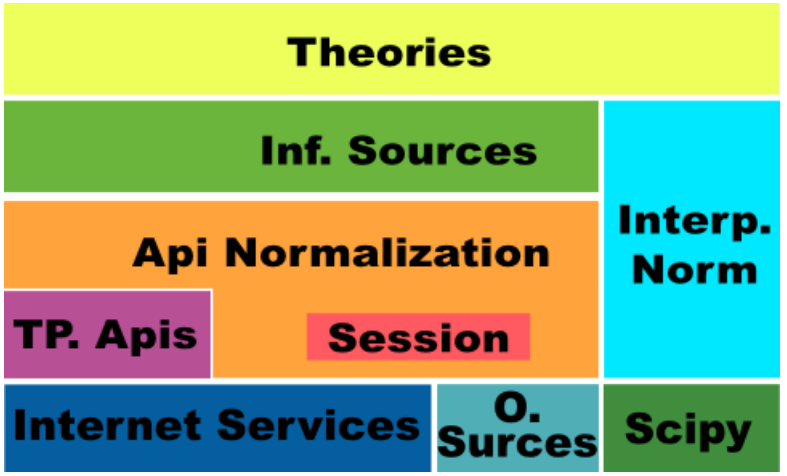
Infopython

Since there is a wide variety of public services that extract statistics and data on new media (web, twitter, etc), including:

- Klout (http://klout.com/)
- Compete (http://www.compete.com/)
- Alexa (http://www.alexa.com/)

To quote a few. **Infopython** focuses on providing a simple API to value through agenda-setting (in the future to implement other theories) to the media regardless of type, using aforementioned services

Architecture:



Analyzing every layer:

- **Internet Service: Corresponds to the different services that exist**

  on the Web for statistics and data mining of new media.

- **Other Sources:** They are other data which feeds **Infopython**, such as databases, excel templates, etc.

- **Scipy: It is an open source library of algorithms and**

  mathematical tools.
  This handles the necessary number crunching.

- **Third Parties Apis:** These are third-party libraries that connect to services that exist in the network. For example:

  - Tweepy used to manipulate data from twitter.
  - Koutpy that connects to Klout

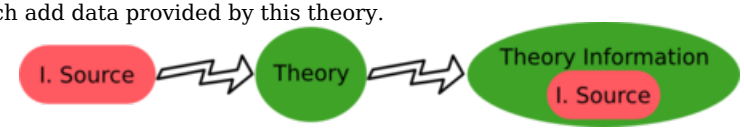
- **Session:** This sub-layer is a module that is responsible for centralizing all necessary settings to access internet services.

- **Interpolation Normalization:** This is a layer of abstraction for different interpolators which has scipy and defines some new all with the same API.

- **API Normalization:** will convert all the answers of all Internet services and third party APIs to common structures (dictionaries) using if needed information contained in the session.

- **Information Sources:** These are the classes that represent our sources. These are connected in a “auto-magical” way to the various standardized API’s.

- **Theories:** This layer has modules that define the behavior and the theory calculations implemented in the **Infopython** (for current version only Agenda-Setting). Each theory encapsulates the media of information in "nodes" which add data provided by this theory.



Now defined all the theory, and the whole architecture, we can mention how is working with the library:

1. **Set up the session:** is to provide the session layer all api key (authentication mechanisms of third party services) required.

Example:

```
from infopython import session

# List all MANDATORY keys from the library
session.NEEDED_KEYS

# set up the session with the keys v0, v1, ...
session.set (v0 = 1, v1 = 2 ...)

# Returns the value of a key
session.get ("v0")

# Delete the session
session.clear ()
```

In the current version all NEEDED_KEY are mandatory and the session is immutable.

2. **Create the media:** Create the media on which you want to check its value. In this version of **Infopython** classes is provided for **2** media:

   - `WebPages:` It represents a web page regardless if it is a twitter profile or blog or whatever. Is suggested as mechanism of audience measurement services of Compete (http://www.compete.com/) or Alexa (http://www.alexa.com/).

     And as a mechanism to measure the impact Page Rank (http://es.wikipedia.org/wiki/PageRank), because if Google says the importance of the information is this, we are not going to argue with Google.

     `WebPage` Api Example:

```
from infopython.isources import webpages

webpages.WebPage google = ("google.com")

google.id # return "google.com"
google.url # return "http://google.com"
google.html # The HTML content of "http://google.com"
google.text # The HTML text of "http://google.com"

google.get_info ("compete") # Compete information of
                            # "Google.com" using the
                            # key of compete provided
                            # in the session
```

   - `TwitterUser:` It represents a Twitter user and not their tweets

     It is suggested as a mechanism for measuring audience the amount of followers, and for impact the information provided by Klout (http://klout.com/)

     `TwitterUser` API Example:

```
from infopython.isources import twitteruser

I = twitteruser.TwitterUser ("leliel12")
yo.id # leliel12
yo.username # leliel12
yo.get_info ("tweepy") # tweepy information of the user
                       # "Leliel12" using the key of
                       # Twitter provided in the session
```

3. **Create Evaluators:** Involves creating **callables** (functions or methods) to receive an information medium as a parameter and return the values shall assume as audience or impact. For example if we decide that our isource `WebPage` draw its **audience** of ** Compete ** and **Impact** by **Pagerank** the functions should be similar to these:

```
# Extract the unique visitors of compete from the WebPage you receive as
# parameter
aud = lambda w: w.get_info ("compete") ["metrics"] ["uv_count"]

# Extract the value of page rank of WebPage that you receive as parameter
imp = lambda w: w.get_info ("pagerank") ["pagerank"]
```

If none of the evaluators were supplied to the agenda, it will try using the supplied interpolators.

4. **Create interpolators:** The interpolators are used as second alternative to extraction of **audience** and **impact**, so each agenda receives 2 interpolators: an audience and an impact interpolator.

Thus the **impact** interpolator will receive as value to interpolate **"X"** to the **audience** and will return a value **"Y"** for the **impact**.

Now, if we wish to interpolate the value of the **Audience** the interpolator will receive as value **"X"** the **Impact** and will return a **"Y"** for the **Audience**.

An example is shown below together.

5. **Create the agenda/s:** When creating agendas they should be provided with different data:

- What kind of information medium will measure.
- A list of media to be measured (optional).
- An audience data extractor (optional).
- A impact data extractor (optional).
- A audience interpolator (optional).
- An impact interpolator (optional).

An example is shown below together.

6. **Evaluate nodes:** The agenda has methods to sort `ISources` by value, only to be iterated and thus generate a ranking of importance for each medium.

By iterating on the `Agenda` it returns various `ASNode` which are data structures that encapsulate the media and add attributes corresponding to **Audience**, **Impact** and **Value** as well as date and time when the node was created.

Full-featured example

```
from infopython import session
from infopython import agenda
from infopython.util import interpolator
from infopython.isources import webpages

# Configure the session.
# All these keys are for mock and for a real test any
# user can register them on the website of each application.
session.set(compete_key = "967b8490-e26a-11df-8cbe-0019662306b1",
            twitter_key = "967b8490-e26a-11df-8cbe-0019662306b1",
            twitter_secret = "967b8490-e26a-11df-8cbe-0019662306b1",
            twitter_user_key = "967b8490-e26a-11df-8cbe-0019662306b1",
            twitter_user_secret = "967b8490-e26a-11df-8cbe-0019662306b1",
            klout_api_key = "967b8490-e26a-11df-8cbe-0019662306b1")

# Create two webpages
google = webpages.WebPage("google.com")
yahoo = webpages.WebPage("yahoo.com")

# Extract measurements
aud = lambda w: w.get_info("compete")["metrics"]["uv_count"] # audience
imp = lambda w: w.get_info("pagerank")["pagerank"] # impact

# an interpolator
itp = interpolator.PieceWisePolynomial([0,0,1,1,2,45,64], [1,3,1,1,2,4,64])

# Create the agenda
# This agenda will try to extract the values of audience and impact with its
# 'Valuators', in case of 'None' will try its interpolators.
# If they return once again None, a value of 0.0 will be returned and medium
# value will be calculated with them.
ag = agenda.AgendaSetting(itype=webpages.WebPage,
                          inf_sources=[google, yahoo],
                          audience_valuator=aud,
                          impact_valuator=imp,
                          impact_interpolator=itp,
                          audience_interpolator=itp)

ag.rank() # sort the agenda by the value of each medium
```

```
# Iterate over each ASNode and print the values of audience and impact.
for i in ag:
    print i.id, "%s + %s = %s" % (i.audience, i.impact, i.value)
```

More Methods of Agenda

Assuming we have an instance, the same agenda from the previous example `ag` and `WebPage`, `google`:

```
ag.value_of (google) # returns the value of google (audience + impact)
ag.impact_of (google) # returns the value of the impact of google
                  # Shall be given what we define as evaluator of
                  # impact would make the call:
                  # Return google.get_info ("pagerank") ["pagerank"]

ag.audience_of (google) # returns the value of audience of google
                # Shall be given what we define as an evaluator of audience
                # would make the call:
                # Return google.get_info ("compete") ["metrics"] ["uv_count"]

ag.wrap (google) # return a ASNode with the values of audience
                # Impact and value of information of google

ag.count (google) # Return how many times this medium is in the agenda

ag.remove (google) # removes first occurrence of google in the agenda

ag.append (google) # add google to the agenda

ag.for_type # would return which type of iSource this agenda was created for
            # WebPage for our example

ag.audience_valuator # None or function calculation of audience

ag.impact_valuator # None or function of calculating impact

ag.audience_interpolator # None or audience interpolator

ag.impact_interpolator # None or impact interpolator
```

Comparing 2 Agendas

In the `Agenda` module there is a feature that is useful for evaluating various agendas with different media.

This function returns a sorted list of `ASNode` for both agendas.

```
from infopython import agenda
from infopython.isources import webpages, twitteruser

# 2 agendas with different media types.
AG1 = agenda.AgendaSetting (iSource = webpages.WebPage)
AG2 = agenda.AgendaSetting (iSource = twitteruser.TwitterUser)

# Iterate over all the media of both agendas
# Sorted by 'value'.
for i in agenda.rank_isources (AG1, AG2)
   print i
```

Final Note: Test

Upon downloading of the library at (http://bitbucket.org/leliel12/infopython/) first thing to do is run the test according to the following steps:

1. **Run**

   ```
   $ python setup.py test
   ```

2. Configure `test.cfg` with the keys of the corresponding API's.

3. **Run now if**

   ```
   $ python setup.py test
   ```

Conclusion

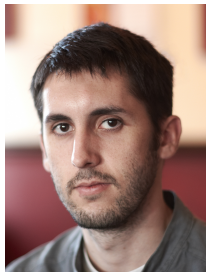
As we saw **Infopython** provides a uniform way of assessing the information. Future versions plan to introduce other types of mass-media since for example, **IMDB** and **GoogleBooks** provides information via API's of traditional media (movies and books) or, going further, **LinkedIn** fairly reliable information about job profiles.

It is also possible integration with natural language processing NLTK or a semantic web tool.

Links:

- Infopython: http://bitbucket.org/leliel12/infopython/
- Theory of Agenda-Setting: http://en.wikipedia.org/wiki/Agenda-setting_theory

How to generate .exe files and installer for a python application



Author: Mariano Guerra From Córdoba, 25 years-old, Systems Engineer degree from UTC Córdoba python programmer and PyAr member for too long ;). emesene creator

This document describes the necessary steps to create an executable file from a python application and how to generate an installer and a portable version for such installation.

This document assumes that the application is GTK-based but should work for other toolkits with some minor changes.

Why an installer?

- avoid having the end-user hand-install a lot of components just for your application
- lots of small installers
- they are hard to find
- the right versions that work well together are even harder to find
- installation in a particular order is required
- praying
- sometimes it may not work despite doing everything right
- easy to automate and document to replicate on each new version
- free the end user of all hassle to actually use your application

Required components

- python
- all libraries used by the application
- py2exe
- nsis
- time and luck

Installers

Here is a list of links to installer for all the components used in the example:

- http://python.org/ftp/python/2.6.6/python-2.6.6.msi
- http://sourceforge.net/projects/py2exe/files/py2exe/0.6.9/py2exe-0.6.9.win32-py2.6.exe/download
- http://ftp.gnome.org/pub/GNOME/binaries/win32/pycairo/1.8/pycairo-1.8.6.win32-py2.6.exe
- http://ftp.gnome.org/pub/GNOME/binaries/win32/pygobject/2.20/pygobject-2.20.0.win32-py2.6.exe
- http://ftp.gnome.org/pub/GNOME/binaries/win32/pygtk/2.16/pygtk-2.16.0+glade.win32-py2.6.exe
- http://sourceforge.net/projects/pywin32/files/pywin32/Build%20214/pywin32-214.win32-py2.6.exe/downloa
- http://sourceforge.net/projects/gtk-win/files/GTK%2B%20Runtime%20Environment/GTK%2B%202.22/gtk2-runtime-2.22.0-2010-
- http://sourceforge.net/projects/gtk-win/files/GTK%2B%20Themes%20Package/2009-09-07/gtk2-themes-2009-09-07-ash
- http://prdownloads.sourceforge.net/nsis/nsis-2.46-setup.exe?download

Installation order

Some installers depend on others, to avoid issues I recommend the following installation order:

- python
- gtk-runtime
- gtk2-themes
- nsis
- pygobject
- pycairo
- pygtk
- pywin32
- py2exe

Extra tasks

- add python’s installation path to the PATH environment variable
- test the installation with a tiny GTK application

```
>>> import gtk
>>> w = gtk.Window()
>>> l = gtk.Label("asd")
>>> w.add(l)
>>> w.show_all()
>>> gtk.main()
```

Test application test

I created a repository with one example application to test the steps that you can find in the following github link:

http://github.com/marianoguerra/PyGtkOnWindows

Steps

- download
- uncompress
- run python setup.py py2exe
- copy the lib and share folders from the GTK's runtime installation (not pyGTK's) to the dist folder
- copy all the files in the dll folder to the dist folder
- remove unused locales and themes from the dist folder (I only leave the MS-Windows theme)
- create the following folder structure: etc/gtk-2.0
- create a file named gtkrc inside that folder with a single line as follows:
    - gtk-theme-name = "MS-Windows"
    - you can change the theme used for another theme that is inside of share/themes and changing the name in the gtkrc file
- right click on ejemplo.nsi y choose "Compile NSIS Script"
- right click on ejemplo-portable.nsi and choose "Compile NSIS Script"
- you should have the installer and portable examples available now
- to make sure that everything works fine, run the installer and the portable version in a Windows installation that didn't have the packets you previously installed

Real application test

Now, to make it more real, we will create an installer and a portable version of a real program. In this case, a personal project called "emesene 2" (http://www.emesene.org/).

Steps

- download it from http://github.com/emesene/emesene
- uncompress
- copy setup.py and ez_setup.py to the emesene folder
- cd emesene
- run python setup.py py2exe
- cd ..
- copy the lib and share folders from the GTK's runtime installation (not pyGTK's) to the dist folder
- copy all the files in the dll folder to the dist folder
- remove unused locales and themes from the dist folder (I only leave the MS-Windows theme)
- create the following folder structure: etc/gtk-2.0
- create a file named gtkrc inside that folder with a single line as follows:
    - gtk-theme-name = "MS-Windows"
    - you can change the theme used for another theme that is inside of share/themes and changing the name in the gtkrc file
- right click on emesene.nsi and choose "Compile NSIS Script"
- right click on emesene-portable.nsi and choose "Compile NSIS Script"
- you should have the installer and portable examples available now
- to make sure that everything works fine, run the installer and the portable version in a Windows installation that didn't have the packets you previously installed

Notes

- some of the required DLLs can be obtained from portable python (http://www.portablepython.com/) and inkscape (http://inkscape.org/)

Memory debugging and defragmentation

Author: Claudio Freire

Introduction

This lecture was planned to be given in 20 minutes. If you can't read it in 20 minutes, reread, reread and re-reread until it takes 20 minutes :-p

Well, actually the text here is only slightly based in the real lecture. I'm one known to improvise in lectures, all I have are the slides as a guide, the general guidelines for the lecture, and this format doesn't lend itself to improvisation. Besides, no freaking way I'm going to remember the exact words I used ;)

Now, really... this is a deep convoluted subject. Reread and re-reread until you get it.

When I started looking for lecture subjects, I decided for memory fragmentation because not long ago I though it was a myth. Ok, yes, it can happen... but, does that happen to anybody? Is it a problem? How naive of me to doubt that could answers could be "yes".

In PyConAr 2010 (and here in this space now), not only I had the chance to transmit around 4 months of picking Python's guts apart, trying to find out what was going on with my applications (where I suffered from memory fragmentation), I also learned that this issue is much, much more common than I'd expected.

When the lecture started, I asked: Does anybody know what memory fragmentation is?. Exactly zero attendants raised their hand. Exactly the amount I had expected. But, in the end, many approached me saying that the same thing that was happening to me (and that I described, and will describe below) was happening to them.

That, according to my post-PyCon estimations, means a 10% of everyone present (assuming they're Python developers) suffer from that issue, and to a 3% of them it means serious operative problems. So the issue was far more relevant than I had expected.

How does Python manage memory?

Before going into this fragmentation thing, we must study some internal details of how memory management works in Python.

How would you say Python manages memory? Malloc? *(a C function that reserves memory for those unfamiliar with C)*

Well, no. Python has unusual and very specific requirements, if it used malloc for all of its memory needs, it would be too inefficient. Python uses, instead, a series of techniques and strategies designed to minimize malloc calls, which are very slow to be used in the core of Python's virtual machine.

- Pools
  - Integer
  - Floating point
- Free lists
  - Tuples
  - Lists
  - Frames *(yes, the frames are objects and need to be managed too)*
- Arenas

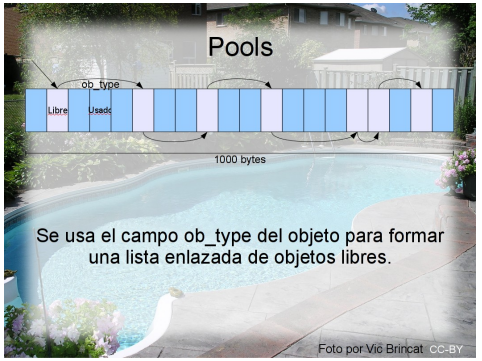
Lets see what each is about, one by one

Pools

Not for swimming.

They're arrays, object vectors of the same type, that Python uses usually to accelerate the creation and destruction of common objects, like integers and floating point numbers.

When python needs one of these objects, it doesn't create one, it creates many (as many as they fit in a pool block), and keeps a linked list of which objects are free within each block.



Structure of an object pool

In a pool, the **ob_type** field (present in all PyObject) is used to link free objects. When a new one is needed, the field is restored (trivial to do), and in general there's no need of further initialization, so it is very fast.

In an object pool, creation and destruction is very fast, and, depending on the type of object, it can save a lot of initialization (in the case of integers and floating point numbers, this is especially true), the objects remain well pack in memory, all close together, and everything works very well.

Free lists

The idea of not having to ask for memory to create or destroy objects that are frequently created and destroyed is something that can be generalized from pools to any type of object (not only those of a particular type), even objects of variable size (where having them all packed in an array is infeasible).

When you do that, you have a free list:

Structure of a free list

Free lists are very similar to a pool, but instead of keeping an array of objects, it simply keeps a linked list of free objects. When an object is destroyed, it can simply go into the free list instead of freeing its memory, to be able to reuse it later.

This idea of having free lists saves tons of calls to malloc, and is particularly useful for strings, lists and tuples, which are variable-sized objects very intensely used in Python, and where a pool wouldn't be practical. They're also the main reason why Python is particularly sensitive to memory fragmentation, and we'll soon see why.

Note also that frames use free lists. Frames are objects, also of variable size (because they need a stack, temporal space for local variables and objects our code will generate), also very intensely used in Python (one gets "created" every time a function call is made). Free lists of frames are a very important optimization (they save a lot of time initializing the frames, a costly operation), but it also contributes to memory fragmentation (as every free list does).

One important thing to remember about Python free lists is that the decision whether to destroy an object or put it into the free list is done at the moment of dereferencing it (when its reference count reaches zero). Once in the free list, it remains there until reused. Python, in order to take this decision, contains a series of limits - X frames, Y 1-sized tuples, Z 2-sized tuples, W strings, etc... (this limit is usually 100 in each case).

Arenas

So... what about the rest? Malloc?

Well, yes and no.

It uses arenas. Which is not spanish for where the glass comes, but a hybrid between pools, free lists and malloc.

For small objects, Python keeps a *list of pools* for each concrete size (remember that pools need objects of the same size, for they're vectors). Each pool has its free list, and each pool block is 8Kb in size. This is called an arena.

For big objects (more than 256 bytes), Python calls malloc directly.

Like Python object sizes grow in 8-byte increments (due to their structure), then there are exactly 32 arenas.

All Python objects are created in this way, even those that use free lists.

Arenas also induce an inner memory fragmentation problem, since no arena block can be freed until all the objects residing in it are freed.

Fragmentation

So, lets see what this memory fragmentation thing is:

Map of fragmented memory

If black is used space, and white is free space, you can see here how there's a lot of free space, but unusable for objects beyond some size, because it's not contiguous space.

Put simply, memory fragmentation happens when there's lots of free space, but it's not contiguous. Like in the above map, much of it is free space, but it's unusable for big objects because, in contrast with a file you can split into several blocks on disk, an object's memory region needs to be contiguous.

So, in contrast with fragmentation in a file system, memory fragmentation makes portions of it unusable. If I needed memory for a big object, say a few megabytes, I would have to use the area towards the end of the map (that is, *extend the virtual image of the process*). This is effectively what malloc does when it's faced with this situation.

The immediate, visible effect here, is an inefficiency in the use of available memory. If my program needed 2GB of memory in theory, it could be reserving 4GB from the operating system (because it has many small pieces reserved that it cannot use). If I have bad luck, this could make my system swap. If I have too much bad luck, it could trash, and die.

Lets see code that fragments memory:

```
>>> l = []
>>> for i in xrange(1,100):
...   ll = [ " " * i * 16 for j in xrange(1000000 / i) ]
...   ll = ll[::2]
...   l.extend(ll)
...   print sum(map(len,l))
...
8000000
16000000
…
792005616
>>>
```

After this, top says:

```
10467 claudiof  20   0 1676m 1.6g 1864 S    0 82.7   1:17.07 python
```

Meaning, even though according to our calculations the program had to consume 800M of memory, it effectively consumes 1.6G. Double that.

Why?

Well the example was specifically tailored to create 50% of unusable holes. The memory is fragmented, then, by 50%.

But there's something worse. Suppose I do:

```
>>> del l
>>> del ll
```

I get from top:

```
10467 claudiof  20   0 1532m 1.5g 1864 S    0 75.6   1:17.96 python
```

If I repeat the fragmentation example, I can confirm that those 1.5G are effectively free for python:

```
10467 claudiof  20   0 1676m 1.6g 1864 S    0 82.8   2:33.39 python
```

But if I try to free them (to the operating system), I can't.

¿WTF?

Enter Guppy

Guppy is a little red fish commonly seen in fish tanks everywhere. Those little fish there, those are called guppy.

Really.

It's also an extension library for Python that contains a module, heapy, which allows me to do memory diagnostics.

Really.

Guppy

So, let's try to use it:

```
>>> from guppy import hpy
>>> hp = hpy()
>>> hp.heap()
```

```
Partition of a set of 23778 objects. Total size = 1760692 bytes.
Index  Count   %     Size   % Cumulative  % Kind (class / dict of class)
    0  10642  45   696652  40    696652  40 str
    1   5432  23   196864  11    893516  51 tuple
    2    345   1   112968   6   1006484  57 dict (no owner)
    3   1546   7   105128   6   1111612  63 types.CodeType
    4     66   0   104592   6   1216204  69 dict of module
    5    174   1    93168   5   1309372  74 dict of type
    6    194   1    86040   5   1395412  79 type
    7   1472   6    82432   5   1477844  84 function
    8    124   1    67552   4   1545396  88 dict of class
    9   1027   4    36972   2   1582368  90 __builtin__.wrapper_descriptor
```

That is, Python (just by launching it) already consumes 1.7MB. In python objects. Heapy doesn't count what is not Python objects, so all that is reported there is memory used directly by python objects.

This means strings, lists, dictionaries, arrays, but **not** mmap objects, or memory used by extension libraries (eg: SDL surfaces in pygame).

Lets diagnose then:

```
>>> l = []
>>> for i in xrange(1,100):
...     …

>>> hp.heap()
Partition of a set of 2612542 objects. Total size = 866405844 bytes.
Index  Count   %     Size   % Cumulative  % Kind (class / dict of class)
    0 2599386  99 854833304  99 854833304  99 str
    1    132   0 10516640   1 865349944 100 list
    2   5433   0   197064   0 865547008 100 tuple
    3    351   0   113784   0 865660792 100 dict (no owner)
    4   1547   0   105196   0 865765988 100 types.CodeType
    5     67   0   105112   0 865871100 100 dict of module
    6    174   0    93168   0 865964268 100 dict of type
    7    194   0    86040   0 866050308 100 type
    8   1472   0    82432   0 866132740 100 function
    9    124   0    67552   0 866200292 100 dict of class
```

So, just as we had calculated, more or less 800M (850M) in Python objects. That's what heapy says.

```
>>> del l
>>> del ll
>>> hp.heap()
Partition of a set of 23844 objects. Total size = 1765236 bytes.
Index  Count   %     Size   % Cumulative  % Kind (class / dict of class)
    0  10690  45   698996  40    698996  40 str
    1   5433  23   197068  11    896064  51 tuple
    2    351   1   113784   6   1009848  57 dict (no owner)
    3   1547   6   105196   6   1115044  63 types.CodeType
    4     67   0   105112   6   1220156  69 dict of module
    5    174   1    93168   5   1313324  74 dict of type
    6    194   1    86040   5   1399364  79 type
    7   1472   6    82432   5   1481796  84 function
    8    124   1    67552   4   1549348  88 dict of class
    9   1027   4    36972   2   1586320  90 __builtin__.wrapper_descriptor
```

¿WTF?

Heapy tells us that Python uses 1.7MB again. Top keeps saying 1.6G. I believe top.

It happens that, in fact, the rest is "free" space (free for Python, not for the operating system).

Doing a differential analysis, we'll get some perspective into the matter:

```
>>> from guppy import hpy
>>> hp = hpy()
>>> heap1 = hp.heap()
>>> # experimento
>>> heap2 = hp.heap()
>>> new_stuff = heap2 - heap1
>>> del l, ll
>>> garbage = heap3 - heap1
```

That results in 3 heap snapshots. *heap1*, as it is after launching Python. *heap2*, after the experiment, and *heap3* after "freeing" everything, and two "differentials", *new_stuff*, what's in heap2 that is new (not in heap1), and *garbage*, what is in *heap3* that is not in *heap1* (that is, what wasn't freed).

```
>>> new_stuff
Partition of a set of 2588725 objects. Total size = 864642976 bytes.
Index  Count   %     Size   % Cumulative  % Kind (class / dict of class)
    0 2588706 100 854134668  99 854134668  99 str
```

```
    1      2   0 10506304   1 864640972 100 list
    2      6   0      816   0 864641788 100 dict (no owner)
    3      2   0      676   0 864642464 100 types.FrameType
    4      2   0      272   0 864642736 100 dict of guppy.etc.Glue.Owner
    5      1   0       68   0 864642804 100 types.CodeType
    6      2   0       64   0 864642868 100 guppy.etc.Glue.Owner
    7      2   0       64   0 864642932 100 tuple
    8      1   0       32   0 864642964 100 exceptions.KeyboardInterrupt
    9      1   0       12   0 864642976 100 int
```

It's worth questioning: Only 850M in strings? And the other 800M to complete the 1.6G?

Well, it happens that the memory looks a bit like gruyere cheese at this time. There's 800M in relatively small strings, but as in each step we'd been freeing half of them (`ll = ll[::2]`), we also have 800M in free but unusable space. Because in each step, also, I need strings a bit bigger than before, so the free holes cannot be reused.

Lets see what happens after dereferencing everything:

```
>>> garbage
Partition of a set of 29 objects. Total size = 2520 bytes.
Index  Count   %     Size   % Cumulative  % Kind (class / dict of class)
    0      6  21      816  32       816  32 dict (no owner)
    1      2   7      748  30      1564  62 types.FrameType
    2     10  34      364  14      1928  77 str
    3      2   7      272  11      2200  87 dict of guppy.etc.Glue.Owner
    4      2   7       80   3      2280  90 __builtin__.weakref
    5      1   3       68   3      2348  93 types.CodeType
    6      2   7       64   3      2412  96 guppy.etc.Glue.Owner
    7      2   7       64   3      2476  98 tuple
    8      1   3       32   1      2508 100 exceptions.KeyboardInterrupt
    9      1   3       12   0      2520 100 int
```

Gotcha!

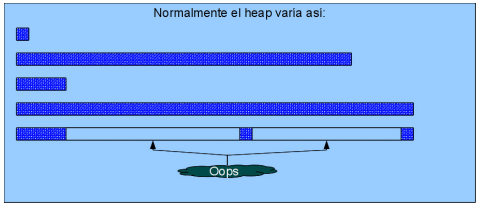
That's important!

These 29 objects stop the heap from shrinking. What reminds me…

The Heap

…of the heap.

Normally, the heap gets bigger and smaller.



Heap lifecycle

> The heap expands and contracts, but in each cycle there can be leftover "garbage", or maybe useful live objects, that prevent it from fully contracting. The memory that is left trapped in the middle cannot be reused by other processes, it's only free to Python.

As you can see on the figure, every time it shrinks, it doesn't completely. Sometimes, objects remain alive in high addresses - since the heap cannot be fragmented (you cannot free space in the middle of the heap, it can only expand or contract), these objects keep memory in the middle reserved for Python. Python can reuse it, but not the rest of the operating system.

This hurts disk cache, other processes (maybe other Python processes, in a webserver it can happen that we have more than one worker running Python), it damages the performance of the whole system.

Guess who have the habit of leaving objects alive in high addresses…

…that's right. Free lists. Here, with guppy, we found 29 objects, probably all kept alive thanks to some free list that keeps them alive. We see a pair of them are Frames, as I said, Frames cause this kind of issues.

We all want to know how to avoid this, so:

Guppy tips

- Don't leave garbage lying around on the floor
    - If you'll be creating lots of small objects, create *persistent* ones first, *transient* ones last.
        - Compiling code (eg: using eval or doing imports) generates permanent strings, called *interned strings*, so compiling on-demand is something to avoid just as well.
        - SQLAlchemy and many other libraries have internal caches, research them and be aware of them and their eviction politics.
- Every time it is possible, prefer a few big objects over many small ones:

        - String lists → comma-separated strings. Or pipe-separated. Or newlines. Whatever.
        - Number lists → `array.array` or `numpy.array`
    - Sweep from time to time
        - If you keep caches with expiration times, clean the cache regularly to remove expired elements. Don't be lazy.
        - Sometimes you can "desfragment" memory, rebuilding persistent structures like caches.

          In fact, java's garbage collector does exactly this, automatically, and many projects are seeking to implement a similar garbage collector for Python, but Python's extension API, Python/C, makes it hard since it allows unmanaged pointers to PyObject, structures that represent Python objects)*.
    - Change is good
        - Don't create eternal structures.
        - Caches always expire.
        - Threads get renewed.
    - The house is for living, the office for working
        - Whenever possible, perform memory intensive tasks in a subprocess, which will free all memory and tidily clean up when finished.
        - The subprocess is the office. You work there.
        - The main process is the house. You live there.

Useful links

- **Slides**: http://python.org.ar/pyar/Charlas#Depuraci.2BAPM-n_y_defragmentaci.2BAPM-n_de_memoria_en_Python (spanish)
    - **How to map memory**: http://python.org.ar/pyar/MapeandoMemoria
    - **Heapy**: http://guppy-pe.sourceforge.net/

NINJA-IDE, an IDE thought for Python

Authors: Diego Sermentero, Santiago Moreno
Both System's Engineering Students that develop free software for the love of it
Website: http://ninja-ide.org
Twitter:

- @diegosarmentero (http://twitter.com/diegosarmentero),
- @f4llen_4ngel (http://twitter.com/F4LLEN_4NGEL),
- @ninja_ide (http://twitter.com/ninja_ide)

How did NINJA-ID start?

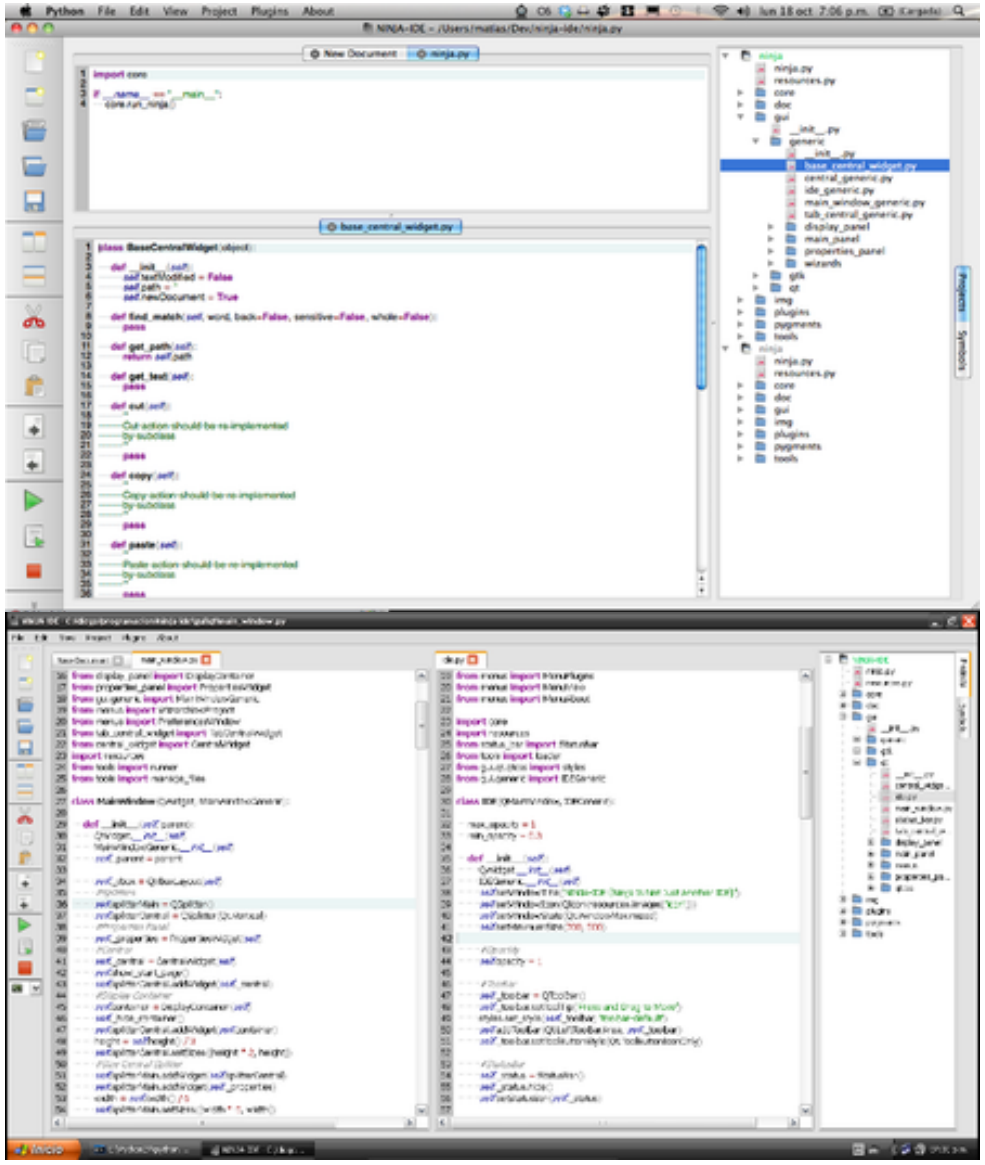
NINJA-IDE was born from a few e-mails sent to PyAr, mails which topic was quite frequent on the mailing list: **"Which is a good IDE for Python?"**, **"Why isn't there a Python IDE that has this or that feature?"** and the responses to those e-mails were always more or less the same as the available IDEs were, in general, not designed for Python just provided the option to integrate it through a plugin so you would end up using a heavy IDE designed for something else where Python support was minimal, or very oriented to a specific framework or not free. So, motivated by the challenge it represented and some interesting ideas laid out in the mailing list we decided to start this project with a single focus: "what features should a good IDE have for a Python programmer?".

With that we started developing NINJA-IDE, its name being a recursive acronym: "Ninja Is Not Just Another IDE". The IDE has been developed for only 7 months now but, thanks to the willpower and programming hours devoted to it, we already have a version 1.1 of it with a lot of functionality. Since its first week of development, NINJA-IDE is used to develop NINJA-IDE helping us find bugs, enhance the application's usability and usefulness through our own experience and continuous use of the tool.

The project is developed under the free GPL3 license and its code can be obtained at http://ninja-ide.googlecode.com

Some of the IDE's current features are:

- Common IDE features for file management, tabs, automatic indentation, editing zoom, etc.
- Being written in Python and PyQt, it is multi-platform and it has been tested in Linux, Mac OSX and Windows systems
- Syntax highlighting for a variety of languages (though centered in Python, syntax highlighting is provided for a variety of other languages for all programmers' benefit)
- Ability to use a Python console within the same IDE
- Integrated project management functionality, recognizing python projects and allowing for the creation of new files and folders within the IDE, automatic "__init__" file creation with the module's information, etc.
- Interface's panels layout is fully and easily customizable, adapting to the user's preferences
- It allows for more than one editor to be visible at the same time, either horizontally or vertically
- Extensible through plugins (which can be created using a NINJA-IDE plugin more easily)
- IDE session support, recovers opened files and projects when the IDE was closed upon opening a new instance
- Auto-complete support (specific to the object being accessed)
- Update notifications
- Lots more!



Who is behind NINJA-IDE?

NINJA-IDE was developed by Santiago Moreno and Diego Sarmentero originally, a week after the beginning of the project they were already developing in it. Thanks to PyAr's mailing lists, blogs, etc. in a very short time the project's publicity made it possible to receive user's bug reports, suggestions on NINJA's mailing list and even code contributions from users and collaborators, some of which became NINJA-IDE committers (like Martín Alderete and Matías Herranz) and others contributors (like Juan Carlos Ojeda and Cesar Vargas).

This strong collaboration and participation we receive allows NINJA-IDE to grow larger and larger every day, making it better and implementing features that its users need. At the same time, comments from NINJA-IDE users motivate us to keep working hard on this tool that we hope it makes it even easier to develop Python applications.

You can get in touch with the project members in multiple ways:

- NINJA-IDE's mailing list: http://groups.google.com/group/ninja-ide
- NINJA-IDE's development mailing list: http://groups.google.com/group/ninja-ide-dev
- NINJA-IDE's IRC channel: http://ninja-ide.org/support.html
- Bug reports: http://code.google.com/p/ninja-ide/issues/list
- NINJA-IDE's Twitter account: http://twitter.com/ninja_ide

We, as NINJA-IDE's developers, are very happy today due to achieving the 3rd place as "Best Free Software" and the 1st place as "Free Software that needs donations the most" (award received as the project with greatest growth expectations) within the contest of *`**Portal Programas <http://www.portalprogramas.com/software-libre/premios/software-libre-donaciones>**`_*, just a few months after starting. At the same time, we are now part of Qt Ambassador's Showcase of Featured Projects.

How do we decide what features to add?

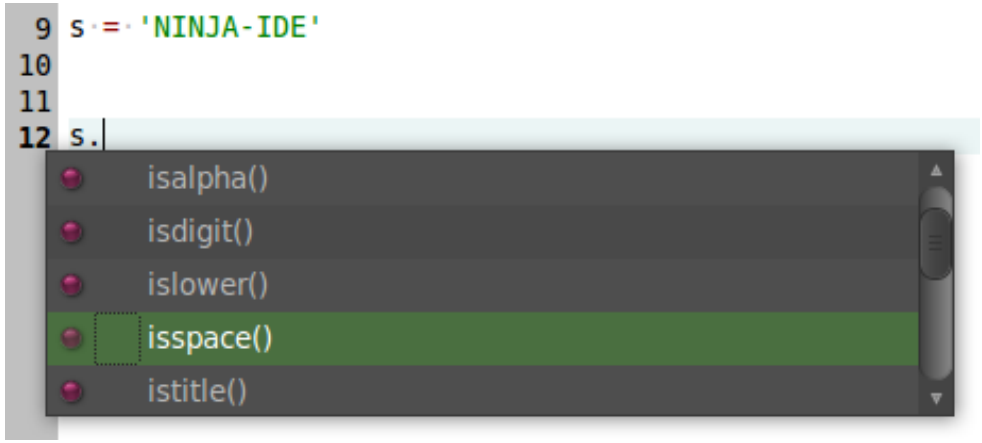
At the beginning of the project we thought of a structure that would allow it to grow and incorporate functionality over time with two main factors as a guide: the code editor and project management. The application was built from the start over these two fundamental pillars, making a good foundation on which to add new features later.

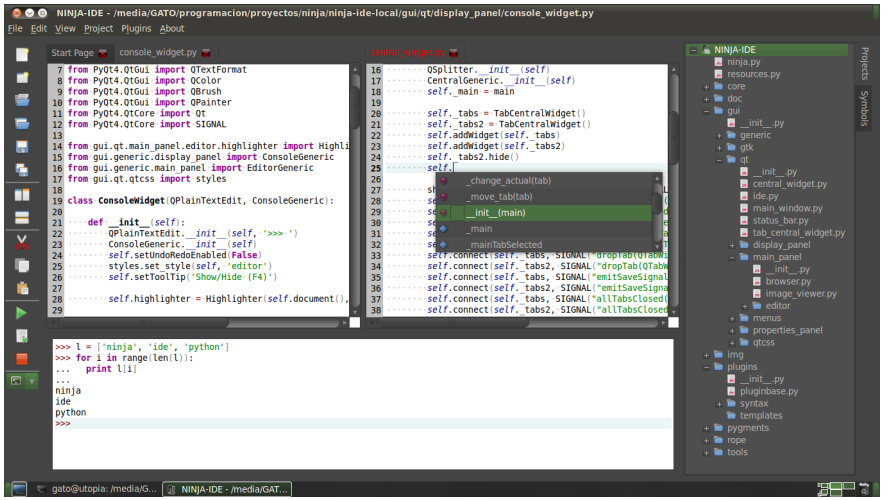
The project went through different stages, starting with a good editor with syntax highlighting, a project and file management next and, later on, features like plugins, autocomplete, session management, etc.

A lot of time Python is seen as the language with most difficulty to provide information about the code that is being written as object inference is not possible while

programming due to dynamic typing, etc. In some cases it is indeed the case that explicit typing would allow for simpler and more detailed analysis, but it is also true that there are a lot of tools and libraries for Python nowadays that help avoid the taboo that is an IDE that can provide real assistance over the code that is being generated. That is why NINJA-IDE tries to provide Python programmers with the same functionality and aides that Java or .NET programmers get on their most popular IDEs. Taking the results and experiences from other language's IDEs, it is our intention to get a Python-centered IDE that satisfies its users.

The mailing list is normally used for the process of suggesting, choosing and incorporating new features into NINJA-IDE, making it a collective decision of all the project's active members. The result is detailed and public information about the feature's objective, the appropriate time to incorporate it and lots of other details. Lots of times these features are motivated by an interesting capability of another IDE, an idea of some of the members or a suggestion from the user's group. This way anyone, be it a user or developer, can express what it wants from NINJA-IDE and we can all decide if it is to be added to the project or would better be implemented as a plugin, at the same time it shows what ideas are being worked on and who are taking control of them to keep the group synchronized.





What do we expect from NINJA-IDE?

NINJA-IDE was born to satisfy what we consider an important need that we saw that other IDEs' approach were not covering. Our intention with this project was to build an IDE focused on Python application development but never forget the need to keep a user community that would allow us to enhance the tool's usability. Nowadays we are very happy to have a NINJA-IDE community as the user's experiences, suggestions and knowledge base it makes development faster and more comprehensive, taking care of details that would be otherwise overlooked.

Future plans

We are now at version 1.1 of NINJA-IDE, codename 'Kunai'. This first version has all the features previously mentioned, providing developers with a practical and robust IDE. Needless to say, as with any project, enhancement and new features to implement will appear.

Some of the features that are planned to be added in NINJA-IDE version 2.0, that we have just began developing, are:

- New extensible and dynamic architecture
- Visual debugger
- Visual browser of a project's modules and classes (based on BlueJ)
- Support for version control tools
- Collaborative edition of a document
- Integrated Qt interface designer

- Code finder
- Virtualenv support
- More refactoring features
- Versioning plugins
- Support for frameworks like
    - Django
    - Google App Engine
- And this is just the beginning!

Release notes for current versions can be found at: http://code.google.com/p/ninja-ide/wiki/ReleaseNotes And a full listing of features proposed for version 2.0 at: http://code.google.com/p/ninja-ide/wiki/Brainstorming_version2

What tools does NINJA-IDE use?

The IDE is developed using PyQt libraries for all GUI work and some functionality. Qt allowed for a solid and highly-customizable interface, making it possible to extend each element as needed to modify its behaviour and adapt it to the IDE's needs.

For syntax highlighting, NINJA-IDE uses its own system based on Qt features that is easily extensible just with a simple JSON file describing the language to add. This methods has several performance benefits, but there is support for Pygments to allow for syntax highlighting that are not included or recognized by this internal system. NINJA-IDE 2.0 is planned to include support for all languages supported by Pygments not only for the speed benefits but also to avoid depending on this external library.

Features like autocomplete, refactoring and those that involve the code itself are based on Rope, a great library and very complete for these kind of situations. Rope allows a Python IDE to have features from IDEs for statically-typed languages.

Code checking using the Pep8 library is also supported nowadays precisely to provide information about how the code adheres to PEP 8 norms.

NINJA-IDE extensibility

NINJA-IDE has a very complete plugin system that allows features to be added just as if they had been native to the IDE. Writing a plugin is very simple and there is even a NINJA-IDE plugin to write NINJA-IDE plugins (recursion anyone?). This "plugin-writing" plugin allows you to choose the portions of the IDE the new plugin will interact with and automatically creates the needed project structure as well, the plugin description file that NINJA-IDE uses as well as the base class for the plugin with the methods that one needs to implement. At the same time, once we are done writing the plugin, it provides functionality to package it so that it can be distributed.

There are 3 NINJA-IDE plugins available at the time:

- Pastebin: sends code to pastebin.con and gets the resulting link back to be able to share the code
- PluginProject: the plugin creator plugin we just talked about
- ClassCompleter: autocompletes some structures while you are writing Python code. For example: creates class constructors automatically making the appropriate calls to parent classes, etc.

For more information about how to develop a plugin for NINJA-IDE you can read the following wiki article: http://code.google.com/p/ninja-ide/wiki/CrearPlugins

Using additional libraries and virtualenv

Author: Tomas Zulberti

The subjects I'm going to talk about are:

- Python's standard library
- What to do when something isn't on that library.
- Installing additional libraries
- Using virtualenv to solve the problems that show up

Python comes with around 350 usable libraries. http://docs.python.org/library/index.html

There's a bit of everything:

- Stuff to work with mails
- Fetching web pages
- Interacting with the operating system
- Working with compressed files
- Parsing different types of documents (xml, json, etc…)

AND MANY MANY MORE….

But first of all, lets see some examples of things you can do with Python's built-in batteries.

Lets send an e-mail using a GMail account:

```
>>> import smtplib
>>> from email.MIMEText import MIMEText

>>> USERNAME = "tzulberti@gmail.com" # Put your username here
>>> PASSWORD = "not-gonna-tell" # Put your password here
>>> mailServer = smtplib.SMTP('smtp.gmail.com',587)
>>> mailServer.ehlo()
>>> mailServer.starttls()
>>> mailServer.ehlo()
>>> mailServer.login(USERNAME, PASSWORD)
>>> msg = MIMEText(" This is the e-mail's content... ")
>>> msg['From'] = USERNAME
>>> msg['To'] = USERNAME
>>> msg['Subject'] = "This is the e-mail's subject"
>>> mailServer.sendmail("tzulberti@gmail.com", "tzulberti@gmail.com", msg.as_string())
```

Lets read a web page's content.

```
>>> from urllib2 import urlopen
>>> response = urlopen("http://python.org.ar/pyar/")
>>> html_content = response.read()
>>> print html_content
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>

        <link rel="icon" href="/images/pyar.ico" type="image/ico">
            <meta http-equiv="Content-Type" content="text/html;charset=utf-8">
            <meta name="keywords" content="Python, PyAr, Python Argentina, user group,
                                           grupo de usuarios, community portal">
            <meta name="robots" content="index,follow">

            <title>Inicio - PyAr - Python Argentina</title>
            <script type="text/javascript" src="/moin_static182/common/js/common.js"></script>
```

Or work with compressed files

```
>>> import zipfile
>>> compressed_file = zipfile.ZipFile("THA_FILE.zip", "r")
>>> for name in compressed_file.namelist():
...     print name
>>> compressed_file.extract()
```

But python's standard library doesn't have it all:

- It doesn't know how to read some file formats (yaml, mp3, video)
- Web Frameworks (pylons, django, web2py)
- Using APIs from Youtube, Deliciosus, etc.…

We've seen that the standard library can do many things, but there are some that are easier if done with third-party libraries:

- Web frameworks save us from some repetitive work.
- API bindings save us the work of reading a lot of documentation on the services they wrap.

- Though python knows how to read text and binary files, it's easier if it simply converts certain file formats to python structures.

Where to look for third-party libraries?

http://pypi.python.org/pypi

There are around 11.000 libraries ready to use. As we saw in Roberto Alsina's lecture (import antigravity), there are libraries to do mostly anything:

- Colored console output
- Alternative interactive python consoles (bpython, ipython)
- Working with weird file formats
- Downloading files from cuevana
- Downloading videos from youtube
- Parsing xml using jQuery-like syntax
- AND LOTS MORE

How do we install packages from PyPi?

First, you need to install python-setuptools.

This depends on the operating system:

- Ubuntu:

```
apt-get install python-setuptools
```

- Windows: you have to download an .exe from here:

http://pypi.python.org/pypi/setuptools/0.6c11#windows

Python packages are generally distributed in an **.egg** file. These files are a kind of zip (with a different extension) that:

- Have the source code
- Have other common files (images, sounds, documentation, etc…)
- Have an all too important metadata files: **setup.py**
- To install **.egg** files you need to install first a module called setuptools.

Once **setuptools** has been installed, you'll have the easy_install command made available.

```
easy_install libraryName
```

For example:

- easy_install django
- easy_install yaml
- easy_install pylons==1.0
- easy_install -U rst2pdf

Something important to note is that the easy_install command also installs all the dependencies. For instance, rst2pdf depends on Pygments and reportlab. If you run:

```
easy_install rst2pdf
```

Then reportlab and Pygments will also be installed. For those using Ubuntu, easy_install works like **apt-get**.

There is an alternative to **easy_install** called **pip**. I'd recommend using that option.

```
easy_install pip
```

Pip can do all that **easy_install** did, and some more:

- Searching in pypi

```
pip search rst2pdf
```

- Uninstalling packages

```
pip uninstall rst2pdf
```

- Listing installed libraries

```
pip freeze
```

- Install all the packages in a file

```
pip freeze > archivoConLibrerias.txt
pip install -r archivoConLibrerias.txt
```

For example:

- pip install django
- pip install -U rst2pdf

However, there are a few problems:

- You have to have admin/root privileges. It's easy when you're developing, but hardly viable in production.
- You cannot install two versions of the same package.

  This is very important when the package isn't backwards-compatible. Some examples are:

  - SQLAlchemy
  - django

  For instance, applications that work wit django 0.9.7 won't work with django 1.2, and viceversa. Because of the way Python's package system works, it only allows one version of any package.

  Therefore, one has to choose which version to use.

Using virtualenv

To solve these problems there is **virtualenv**. It takes care of creating an isolated Python environment within the system. First, you have to install it:

```
easy_install virtualenv
```

This is the only time you'll need to be root/admin to install a library. All the servers that allow hosting python projects have it preinstalled.

If you don't have root/administrator privileges, you can also use it without installing it into the system. For this, you can download the tarball from:

http://pypi.python.org/pypi/virtualenv

Once installed, the first thing to do is to create a virtual environment:

```
virtualenv environmentName
```

This will create a folder **environmentName** in the directory we're sitting. If you didn't install it into the system, if you downloaded the tarball, then use

```
python virtualenv.py environmentName
```

The previous command created a Python environment that is separate from the system's global configuration. Therefore, one does not have any permission problems there because one owns the tree.

To tell the operating system that we want to use the environment we just created, and not the system's, it is necessary to activate it.

Linux:

```
source environmentName/bin/activate
```

Windows:

```
environmentName\Scripts\activate.bat
```

Both in windows and in linux, when you activate an environment, you'll see the name of the activated environment between parenthesis:

```
(environmentName)tzulberti@myhost:~
```

When you have an active environment, you can see that the python you're using is not the global one, but the one inside a folder **virtualenv** created.

```
(environmentName)tzulberti@myhost:which python
/home/tzulberti/environmentName/bin/python
```

The same happens with **easy_install** and **pip**. Also, any packages installed will be installed within the virtual environment.

To stop using the environment we have to do the following:

Linux:

```
deactive
```

Windows:

```
environmentName\Scripts\deactivate.bat
```

How does virtualenv work?

When one creates an environment, inside it three folders are created:

- bin: when one has the environment active, and executes a command like *python*, *pip* or *easy_install*, one of the binaries in this folder is ran, instead of the system's.
- include: it's simply a link to the files installed with Python.
- lib: this is another important folder. Just like **include** it has a folder named python, but in contrast to **include**, this one isn't a link to the files installed with Python. Instead, the folder has two important things:
    - A link to some Python files. In this case, the .py
    - A folder, **site-packages**, which is where packages are *installed* when one uses **pip** or **easy_install**

Q & A session (Part 1)

Before I commented that one of the problems we had was that we couldn't install two versions of the same library. What I never did is comment how this problem is solved.

The solution is easy. You create a virtual environment to use with django 0.97, and another for 1.2. You can create as many virtual environments as you wish, and they're isolated from each other.

For production servers, there are configurations that can be set up in apache for it to use some specific virtual environment.

Q & A session (Part 2)

I create a virtualenv, activate it, and do **pip freeze**, and lots of things that I never installed appear in the environment.

The default behavior of virtualenv when one creates an environment is to also create it with the packages one has installed in the system. For it not to do this, when you create the environment, you can do:

```
virtualenv --no-site-packages environmentName
```

This makes it create a completely empty environment. Lastly, another option to note is **—python**, which makes the virtual environment use that version of python (it is necessary to have installed that version for this to work). For instance, in my machine I have both Python 2.7 and 3.1, being 2.7 the default one.

Thus, when I create an environment, it will be created with Python 2.7. For it to use Python 3.1, I have to do the following:

```
virtualenv --python=python3.1 environmentName
```

Q & A session (Part 3)

There are some packages that have code written in **C**. What happens with those?

Well, in those cases pip or easy_install will try to build the **C** code when the package is installed. In distros like Ubuntu, I recommend installing:

- build-essential
- python-dev

Those two packages make it much easier to build.

However, in Windows it is not so easy. Several packages that have **C** code are distributed as an **.exe** too. However, those installers do not allow us to select where to install them. Here `http://www.developerzen.com/2010/09/23/the-complete-guide-to-setting-up-python-develo` is a very good guide on how to make it build packages as it downloads it.

Q & A session (Part 4)

Do you recommend installing system packages or is it better to use easy_install?

There are various distros that come with various python packages that can be installed in the system. For example:

```
apt-get python-numpy python-reportlab python-pil
```

But the versions that can be installed with **apt-get** are old. Here we'll see some comparative examples:

Package name	Ubuntu 10.10 version	PyPi version
rst2pdf	0.14.2	0.16
reportlab	2.4.3	2.5
django	1.2.3	1.2.3

Thus, it depends on what one wants to do. For instance, if one wants to make a program work with Ubuntu, then one can create a virtual environment and install the same

versions that **apt-get** has, or directly install system-supplied ones.

What I would **NOT** recommend is overwriting the installed version. For instance, if one does:

```
apt-get python-reportlab
```

Then you do NOT do:

```
easy_install -U reportlab
```

Not without using a virtual environment, because it could break the system.

PET Challenge

Authors: Juanjo Conti y Alejandro Santos

In the previous PET issue we presented our first challenge: provide the shortest program capable of factoring a number (taking some rules into account). We received a lot or responses, the first ones around 500-characters long, the last ones squeezing every byte out of the code to get the shortest solution.

From the 58 participants we had, we are proud to tell that we had 3 winners. What's more, their 111-character long solutions get the same results in different ways, cutting, massaging and perfecting their original solutions.

The winners

Without further ado, the winners and their solutions are:

Chema Cortes with pet1-pych3m4.py:

```
n=input();d=1;r=""
while d<n:
 d+=1;s=0
 while n%d<1:n/=d;s+=1
 if s:r+=" x %d"%d+"^%d"%s*(s>1)
print r[3:]or n
```

Javier Mansilla with pet1-jmansilla.py :

```
s=input();_='';i=1
while s>i:
 c=0;i+=1
 while 1>s%i:c+=1;s/=i
 if c:_+=' x '+`i`+'^%i'%c*(c>1)
```

Oswaldo Hernández with pet1-hdzos.py:

```
n=input()
d,f="",1
while n>1:
 f+=1;r=0
 while n%f<1:r+=1;n/=f
 if r:d+=" x %s"%f+"^%s"%r*(r>1)
```

Special mention

In this opportunity we have considered awarding the "most cheating" award to Darni, that asked us not to rename his file "pet1-darni.py" but use his artistic pseudonym: *ZT1pbnB1dCgpCnM9JycKZD0xCndoaWxlIGU+MToKCXA9MDtkKz0xCgl3aGlsZSBlJWQ8MTplLz1kO*

So this is the content of the file pet1-ZT1pbnB1dCgpCnM9JycKZD0xCndoaWxlIGU+MToKCXA9MDtkKz0xCgl3aGlsZSBlJWQ8MTplL

```
import base64;exec(base64.decodestring(__file__[5:-3]))
```

Additional information about the competition can be found at http://python.org.ar/pyar/Proyectos/RevistaPythonComunidad/PET1/Desafio

New challenge

This new challenge was written by Alejandro Santos, who loves problem solving and spent the last week sending several puzzles to the PyAr list.

Thanks to Matías "Tuute" Bellone for his revision of the problem's first draft.

The QQQ number

QQQ is the most popular social network of the Ainohtyp country. Microblogging-style QQQ is a social network where users can only post the word "QQQ". The users can also have other users as contacts, but the other user has to do so too.

The network's administrators built a list with the user sorted by the amount of contact. Curly is at the top of the list with the most contacts.

Your job is to create a program that finds the minimum distance between each user and Curly. That is, the minimum amount of hops between Curly and the rest of the users. If there is more than one way, the shortest is chosen.

For example, lets say Curly is friends with Larry and Moe, Larry is friends with Sue and Sue is friends with Curly. Sue has a distance of 1 with Curly and Larry and Moe have a distance of 1 as well.

It is possible to get from Curly to Sue through Larry, but as Sue is friends with Curly the minimum distance is 1. But if Howard is friends with Moe then the minimum distance from Curly to Howard is 2 as you need to jump through Moe.

Input data

All data must be read through standard input.

Due to privacy concerns the data will be anonymized, you will only get an ID of each user.

The first line will have two numbers: N and M with the amount of users and the amount of friendships in the network respectively.

The users are numbered consecutively, without skipping any user, starting with 1. Curly is number 1. In the following M lines there will be two numbers indicating a relationship between two users.

Some users have no contacts or have no way to build a chain of contacts back to Curly.

Output

You need to print, through standard output N lines with two numbers separated by a single white space. The first number will be the user's id and the second number will be the minimum distance to Curly.

If there is no way to calculate that distance, the second number should be the character "X" instead.

Example

Input:

```
8 7
1 2
1 3
1 4
2 5
3 5
5 6
4 6
7 8
```

Output:

```
1 0
2 1
3 1
4 1
5 2
6 2
7 X
8 X
```

Participants must send their solution as a .py file named pet2-USERNAME.py (replacing "USERNAME" with a username of your choice) and it will be run with python 2.7 in Alejandro's computer. The winner will be the one to get the solution that passes all specially-crafted tests in the shortest time.

Send your solution to revistapyar@netmanagers.com.ar with `DESAFIO2` as the subject of the mail before 2011-July-31.

Good luck and happy brain squshing!

Clarifications and feedback

Any clarifications, if needed, and feedback for participants will be provided through PyAr's wiki at: http://python.org.ar/pyar/Proyectos/RevistaPythonComunidad/PET2/Desafio