PET Python Entre Todos

La revista de la comunidad Python Argentina

Número 2
Abril 2011

Especial Pycon Argentina 2010

ISSN 1853-2071

http://revista.python.org.ar

Licencia

En Este Número

Staff

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

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

PET es la revista de PyAr, el grupo de usuarios de Python Argentina. Para aprender sobre PyAr, visite su sitio: http://python.org.ar

La foto de la tapa, hecha por Juan Manuel Costa Madrid jmcosta8@gmail.com, se la puede encontrar acá: http://www.imgs.com.ar/imgs/2/9/b/29b7bcefc7ab3eee01c8e5f5458fefc2ffcfd4a7.html licencia CC-by-sa.

Editor responsable: Roberto Alsina, Don Bosco 146 Dto 2, San Isidro, Argentina.

ISSN: 1853-2071

¿Qué es PyConAr?

Autor: Tomas Zulberti y Juan B Cabral

Estaba editando los artículos para que salga esta versión especial de la revista y me di cuenta de algo: Salvo para los que fueron, nadie sabe que es PyConAr, así que mediante este articulo voy a intentar de darles una idea.

PyConAr es un evento en la cual se dan diferentes charlas sobre temas relacionados con el lenguaje de programación Python, y su nombre proviene de una conferencia homónima que se viene haciendo en realizando en Estados Unidos regularmente la cual se llama “PyCon” a secas. Si bien el evento Americano se realiza hace varios años, nuestro primer PyConAr fue el que se realizo Septiembre del 2009 en la Ciudad Autónoma de Buenos Aires.

Ahora… ¿Qué fue PyConAr 2010?

El año pasado la conferencia nos cito en Córdoba, en la universidad Empresarial Siglo XXI. La misma estuvo compuesta de 4 tipos de eventos:

- Charlas: Fueron disertaciones de 20 minutos acompañadas 5 minutos para que los oyentes realicen preguntas. Durante el evento se realizaban 3 charlas al mismo tiempo en 3 aulas diferentes; y entre grupo y grupo de charlas había 15 minutos de recreo por si uno quería quedarse hablando con el disertante. El total de charlas en PyConAr 2010 fue de 43.
- Keynotes: En las Charlas Plenarias (o keynotes), todos los asistentes del evento se juntaron en un auditorio de gran capacidad para las exposiciones de invitados importantes. En el evento hubo 3 de estas charlas (1 el viernes y 2 el sábado) al momento del cierre cada jornada.
- Charlas relámpago: Estas se organizan durante el el mismo día de la conferencia. Las dos grandes diferencias sobre las chalas comunes son:
    - Tienen que durar 5 minutos, y no hay tiempo para preguntas
    - No tiene porque estar relacionada con Python. Se puede dar una charla de cualquier tema.
- Posters: Fue la primer PyConAr en la que hubo posters. Los mismos son un resumen en forma de Poster (sic) de un proyecto y lo estos se exponen en algún lugar de mucho transito de asistentes para que los lean en sus recreos.

Pero la conferencia no es solo sobre Python. También es sobre lo humano; es una oportunidad de vernos las caras con las personas que te contestan un mail por la lista de correo o una charla en el IRC.

PyConAr es un evento de laburo, de diversión, de camaradería, de amistad y quien sabe si alguna pareja no se forma por ahí. Pero así es PyAr, un grupo de locos que aman lo que hacen y producen cosas además de solo divertirse. Gracias a todos por hacer de PyConAr una realidad y gracias PyAr por existir y darle a mucha gente un lugar para compartir su pasión.

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/

Cómo se hizo PyConAr2010

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

alias perrito666, con contribuciones anónimas en el Wiki de PyAr.

Hitchhiker’s guide to make a PyConAr2010

La PyConAr 2010 Se fue como vino, todos parecen haber terminado muy felices. Estos casi 10 meses de laburo me dejaron pensando en lo terriblemente mas fácil que hubiese sido este evento si hubiese sabido las cosas que se ahora. Así que, en pro de la próxima persona que tenga que hacer esta ardua pero noble tarea, acá viene “El tutorial de PyCon”

Antes que nada un consejo, si no valoras un momento único en la comunidad y el hecho de crear un espacio enriquecedor para que la gente comparta mas que cosas como el dinero y la comodidad, quizás quieras pensar dos veces antes de comenzar con este evento. De seguro vas a perder mucho tiempo de laburo y probablemente algo de plata y el último mes vas a dormir muy poco y tener que renunciar a tu salud estomacal y tiempo de esparcimiento.

El tutorial solo tiene la parte organizativa y los detallitos del evento, la parte técnica la vamos a ir resolviendo juntos durante estos meses y anotando a medida, mucho del soft que hay para generar cosas necesita ser reescrito, también te voy a ayudar en eso.

Elección del equipo

Antes de pensar en marchandising, locales, temáticas y demás yerbas, vas a tener que pensar en un equipo, el pilar fundamental de cada evento. Mas allá de la gente necesaria para llevar a cabo el evento en el día, vas a necesitar un equipo que este dispuesto a ir con vos a cualquier lado y a poner la misma cantidad de garra durante varios meses.

En mi experiencia, necesitas:

- 2 personas para el marchandising y la impresión de gráficas.
- 1 persona para el diseño (suponiendo que tiene una base con la cual trabajar en que le indicas que tipo de gráfica hace falta y los datos que lleva).
- 1 administrador de sistemas (si usas un sistema, en nuestro caso PyConAr, necesitas alguien que se encargue de cada vez que explota)
- 1 webmaster (alguien que cargue contenido en el sitio, vas a necesitar proveer mucha información y no tenes tiempo de buscarla y procesarla vos)
- 1 community manager y encargado de prensa (suena pavo para un evento, pero lleva mucho tiempo hacer difusión por cuanta red social anda dando vuelta)
- 1 tesorero/contador/malabarista (estas cosas mueven mucha guita, al final del día querés cuentas claras)
- 1 encargado de sponsors (es confuso para las empresas que no haya 1 persona univoca con la que contactarse, un sponsoring tiene que manejarse con la persona con la que se hizo el primer contacto)
- 1 responsable de negociar la venida de los keynotes.
- 1 coordinador de revisores de charlas.

Algunos tips para tener en cuenta:

- Un amigo puede ser una luz, pero no necesariamente un buen colaborador: Cuando tenes un colaborador, tenes que poder delegar una tarea y suponer de ahí en adelante que esta se realizara en tiempo y forma.
    - Delegar no es desentenderse:
        - Tenes que tener en cuenta que si surge un problema tenes que estar listo para darle una mano a tu colaborador para volver a ponerse en carrera.
        - Tenes que pasar toda la data necesaria para que tus colaboradores puedan realizar su tarea y tomar decisiones inteligentes.
        - Si vas a delegar una tarea que requiere recursos o decisiones mas allá del rol de la persona, tenes que estar disponible para contestar a sus inquietudes.
- Entusiasmo no es habilidad ni tiempo: Si bien cuando uno decide hacer un evento de magnitud considerable y muy interesante tenemos muchos prospectos de colaboradores, no todos son candidatos a coordinar tareas.
    - Muchos tienen mas entusiasmo que tiempo y no comprenden las consecuencias de incumplir alguna de sus tareas asignadas, a fin de cuenta uno suele ser el que pone la cara.

    - A veces la disposición y responsabilidad no vienen acompañados de la habilidad de tomar las decisiones necesarias al momento de actuar. En este caso delegar nos puede resultar contraproducente, puesto que terminaremos pasando mas tiempo respondiendo preguntas y tomando decisiones que si hubiésemos hecho la tarea en un principio.
- Juventud e inexperiencia no es sinónimo de inmadurez e inhabilidad: Los jóvenes de tu grupo muchas veces están mas en contacto con la realidad actual que vos y tienen tiempos mas libres o acomodables que aquellos que ya trabajan full time y tienen familia u ocupaciones.

A fin de cuentas elegir gente no es tan difícil, solo tené en cuenta las opiniones de tu comunidad local y averigua sobre experiencias pasadas de los demás, la base de nuestra comunidad es la reutilización de la experiencia pasada.

Si te quedan dudas luego de todo esto, podes probar con alguna tarea pequeñas para asegurarte de que tus impresiones sean las correctas.

Lugar

No es simple elegir un lugar, sobre todo cuando no sabemos que cantidad de personas vamos a alojar. Debemos tratar de elegir un lugar donde entre la máxima gente que podría llegar a venir al evento, mirar eventos de características similares del pasado ayuda bastante. Medir contra eventos en Capital no es una buena idea, a menos que hagas un evento en Capital, siempre llevan mas gente. Si hay disponibles números de alguna versión en el interior mas vale usa esos para calcular. Otra cosa muy importante a tener en cuenta es que tan difícil/costoso es poner internet en ese lugar. Vas a necesitar pensar en:

- Posibilidad de tender cableado: Distribuir internet para un par de cientos de personas es bastante difícil, usualmente necesitas tender cableado para llevar los APs a todos lados, tené en cuenta cuando negocies el lugar que sea posible utilizar cableado pre-existente o poner uno nuevo. Para saber que necesitas consulta con USLA quienes tienen la “caja de maravillas” unos 70kg de equipamiento para red, infraestructura eléctrica y otras cosas necesarias para eventos.
- Posibilidad de contratar un ISP: La cantidad de personas que vas a tener es raramente controlable en cuestiones de red (mas allá del control que tu dispositivo de routeo aplique) por eso vas a necesitar un buen caño de internet. Asegurate que algún ISP pueda proveer esta conectividad durante el evento, la primera opción es intentar que el ISP del lugar del evento, si es que lo hay, agrande la capacidad del servicio por unos días. Trata de que no sea un servicio hogareño, estos tienden a no aguantar.
- Posibilidad de cargar la infraestructura eléctrica: Lo mas probable es que vayas a enchufar varias zapatillas con notebooks. Asegurate que los tableros eléctricos del lugar no vayan a volar por los aires cuando le conectes un montón de notebooks, proyectores, equipos de amplificación de audio, APs, switches, etc. Si podes trata de llevar tus propios dispositivos de protección eléctrica.
- Accesibilidad: tené en cuenta que las comunidades de software tienden a ser muy diversas, averigua si el edificio, al menos en las partes que vas a usar, esta preparado para recibir personas con discapacidades motrices sin ayuda de terceros. Adicionalmente asegurate de que sea accesibilidad-friendly el acceso a la zona, de nada nos sirve un ascensor para sillas de ruedas que sirva para subir las escaleras en un templo en la punta del himalaya.
- Habilitación: Corroborar si hace falta alguna habilitación especial para el evento, algún trámite municipal, etc.
- Seguridad: Si queremos volver a organizar otros eventos después del que estamos planeando, debemos asegurarnos de poder hacerlo en libertad, hacerlo detrás de rejas suele traer inconvenientes... léase: nos hacemos responsables del evento (y sus posibles consecuencias hacia terceros).
    - Seguridad Eléctrica: Muchos locales no tienen la instalación preparada para que ”usuarios particulares” enchufen dispositivos a la red eléctrica, tienen instalación industrial con otros requisitos de seguridad. Asegurarse de que los tomas de uso público cuentan con protección diferencial. El encargado de la sala debe saber dónde y cómo cortar la energía en caso de accidente. Sería una buena medida hacer figurar en el contrato del local que la instalación cuenta con la debida protección (para deslindar responsabilidades)
    - Prevención de accidentes: Se debe tener especial cuidado sobre todo cuando hay escaleras, no debe haber cables y objetos que puedan ser causa de tropiezos; alambres sueltos, etc. .
    - Plan de evacuación: En muchos lugares es obligatorio tener pegados en lugares visibles el sentido de circulación para evacuar el recinto, además se deben identificar las salidas de emergencia. Bomberos voluntarios nos pueden ayudar desinteresadamente en esto.Una vez elegido el lugar y confirmado, asegurate de que lo que te prometan quede escrito y registrado en todos los medios que ellos consideren necesarios y oficiales, es importante en este caso atender a las necesidades de la burocracia. Registro de los acuerdos y respeto de los protocolos del lugar ayudan a sobrevivir cambios de autoridades y otros imprevistos.

Sponsoring

Suponiendo que ya elegiste un lugar, una fecha y tenes un equipo seguro que te ayuda, es hora de ir a buscar dinero. Es la costumbre de las comunidades de software libre de Argentina que los eventos no tienen costo alguno. Esto es bueno del punto de vista de las bajas barreras de entrada, pero causa muchísimos problemas en otros frentes, trato de contarte mi experiencia y la que recogí de la gente de PyConBr para que puedas enfrentarlos.

- Un evento sin entrada no implica un compromiso por parte de muchos de los potenciales asistentes: preparate para cantidades de inscriptos bastante superiores al número de asistentes. El problema de esto es que uno suele tener que planear varios factores, que muchas veces incluyen invertir dinero, en función de la cantidad de asistentes. En un evento pago la no asistencia no implica un problema tan grave, ya que el pago se realizó y esto cubre al menos una parte del costo asumido por esa persona. Intentá que la inscripción sea detallada y que quede en claro para la gente que su inscripción no es solo un acto de apoyo, que desata una serie de acciones de tu parte. Adicionalmente intentá comunicarte con los asistentes regularmente para confirmar asistencia.

    - "'tip:"' la gente de PloneConf2010 creó una lista de correos que incluía todos los asistentes, es interesante para que compartan expectativas sobre el evento, que organicen entre ellos cuestiones logísticas y para enviar anuncios generales y que estos se discutan, sin contar el buen feedback que provee.

    - Un evento sin entrada no proyecta seriedad a las empresas: Esto puede costarte algunos sponsors, especialmente entre las que son mas "vieja escuela" que ven un evento como un lugar de recruiting y capacitación. Por el lado del recruiting no les parece que tu barrera de entrada sea lo suficientemente alta como para serles útil. Por el lado de la capacitación, no creen que con tan poco costo tu curso sea del suficientemente alto nivel.

    - Cualquiera puede acceder: Si bien pedimos un formulario bastante complejo para entrar, siempre terminamos permitiendo el acceso en la entrada. Debido al volumen de gente intentando acreditarse no solemos pedir muchos datos a los inscriptos in situ. Si te es posible hacé esto en un lugar donde puedas restringir la entrada al máximo a gente que solo tenga identificación del evento. Cuando llegue alguien que no se registró, ponelo en alguna cola de espera hasta que hayas acreditado a los pre-inscriptos. Toma todos los datos necesarios de la gente no registrada, tenes un compromiso de ofrecer un lugar lo mas seguro/cómodo posible para que la gente pueda desarrollar su actividad de comunidad lo mas libremente posible. Si tenes la capacidad humana pone una persona específica a registrar gente imprevista y alguien chequeando la entrada de gente desconocida.

    - Si te es posible podes implementar un sistema de entradas mixto, esto debería complacer a ambas partes: empresas y solo comunidad. Para la gente que quiera registrarse como asistentes de empresas podes implementar un arancel, a cambio en el badge, distinto para los asistentes enviados por empresas, va el nombre de la misma. Podes agregar cosas que lo hagan mas atractivo, como un badge personalizado si envían mas de N personas o una cantidad de entradas empresariales si compran un sponsoring. Recordá siempre que el que faciliten que vaya gente es tan importante como que compren un sponsoring, la gente hace a los eventos interesantes. Si conseguís que una empresa envié a sus developers además tenes el auspicio implícito de la misma, ya tenes algo para mostrar a las demás. En cuanto a las personas de la comunidad o que van como individuos (aunque sean de una empresa) la entrada es claramente gratuita. No he implementado esto personalmente, pero vale la pena explorarlo y extender la idea.

El segundo asunto importante del Sponsoring es la venta de los planes de sponsoring. Luego de una larga conversación con Dornelles Tremea y Érico Andrei sobre la búsqueda de auspiciantes, encontramos algunos items que vale la pena compartir.

En lo que respecta al nivel de costos hay dos tipos de empresa:

- La pyme relacionada al FLOSS o la tecnología de tu evento: Estos le ponen el pecho a las balas, están totalmente a favor de lo que haces y te dan una mano mas porque comparten tu ideología que porque tengan interes en publicidad (que tambien lo tienen). Muchas veces no pueden deshacerse de una cantidad alta de dinero, pero si les interesa ayudarte. Esta bueno que tengas un pricing especial con ellos, podes arreglar también intercambios con ellos, quizás tengan algo que vos necesitas además de dinero. Siempre recordá aclararles el porque de este plan de pricing, que entiendan que implica un cierto compromiso con la causa que el sponsoring puramente comercial no.

- La empresa grande: Para esta empresa, aunque suene difícil de creer, mientras mas caro mas atractivo. Hay una componente importante de circo en estas cosas, quieren poder mostrar que ellos están a la altura de la situación y a la cabeza de eventos mainstream, su forma de medir estos es :

    - cantidad de asistentes.

    - costos (del sponsoring y de las entradas, como hablamos antes)

Esto nos lleva al tercer asunto importante del sponsoring: Para muchos de los sponsors tu evento no es lo mismo que para vos, muchas veces no comparten la carga ideológica. El evento es un producto, tiene mas que ver para ellos con publicidad, imagen, recruiting y otras yerbas comerciales. Como todo producto, para venderlo, necesitas un vendedor. Conseguite una persona con habilidades de venta de verdad, de ser posible, profesional. Este es uno de los puntos en los que creo que es mas que válido, incluso crítico, invertir dinero. Podes arreglar algún porcentage de lo recaudado a cambio de los servicios. La persona que tratara con los sponsors deberá entender los códigos que estos manejan, saber leer las necesidades y anticipar que paquetes serán atractivos para cada uno de los posibles inversores. Se que pensas que la comunidad todo lo puede, pero te prometo que no. Para tratar con un área comercial o de RRPP no sirve un técnico, esta gente no compra un producto, compra al vendedor. Luego de comprobar la experiencia real de la gente de PyConBr (ellos tercerizan casi todo el evento luego de muchos años) entiendo que un vendedor con experiencia puede darte mucho mas alcance y visibilidad. Mientras mas llegada y sponsors tengas, mejor sera la calidad del evento que puedas organizar para la comunidad, pensá en pasajes de disertantes interesantes que no pueden costear el viaje, becas, material informativo, comida, café, todas estas cosas están buenas en un evento y cuestan plata.

Finalmente un consejo: tratá de equiparar tus precios a eventos similares de otros lugares, en caso de la PyConAr, seria bueno que los precios y prestaciones de los sponsores, mientras que no sean disrruptivo al espíritu del evento, estén equilibrados (salvando las diferencias económicas obvias de ambos paises).

Información

Este fue quizás nuestro punto mas flojo durante el evento del 2010. No debemos subestimar lo importante que es la información para la gente que viene del exterior y necesita calcular gastos, tiempos y logística en general. Idealmente para el momento en que confirmaste un lugar y vas a hacer un llamado oficial, vas a necesitar proveer, de la forma mas clara y completa posible, los siguientes datos:

- Dirección del lugar: La dirección de la sede del evento, acompañada de un mapa, de ser posible de algún sistema de mapping del tipo de google maps u openstreetmap, para que la gente pueda jugar con eso y familiarizarse con el lugar.
    - Landmarks: Referencias al lugar desde distintos hitos de la ciudad (monumentos, shopping centers, universidades, etc), muchos pueden tener una idea de alguno de estos lugares y esto los ayuda a ubicarse mejor.
    - Transporte público urbano: Todo tipo de transporte urbano que pase moderadamente cerca, el nombre de la parada y como llegar desde la misma al punto exacto del evento. Siempre esta bueno aclarar cual de estos medios llega desde Aeropuertos, terminales de ómnibus, puertos, etc. de donde podrían estar mas interesados en llegar los visitantes de otras ciudades. No te olvides del precio y medio de pago de los transportes urbanos.
    - Transporte interurbano: Como llegar al menos de las ciudades mas importantes del país a la tuya, estaría bueno al menos un transporte de cada provincia. No existe algo como demasiado detallado, mientras mas información mejor. Es muy importantes que para todos estos transportes también incluyas el precio.
    - Internacionales: Como se llega a tu ciudad de otros países, si contas con algún tipo de arribo internacional aclaralo. La ciudad mas cercana a la que se puede llegar en avión u otro transporte internacional (esta de mas decir que también pongas algo de como llegar desde esta ciudad a la tuya, un link a interurbano al menos)
    - Estacionamientos: Mucha gente me preguntó si podían estacionar cerca en el último evento, trata de relevar los estacionamientos mas cercanos y pone algo de información de donde están junto con el precio de los mismos y los planes (por hora, jornales, semanales, etc)
- Alojamiento: Mucha gente viene de afuera y prefiere la palabra de un local a la hora de elegir alojamiento. Como local probablemente no sepas mucho de alojamiento, después de todo ya vivís ahí, no hay muchos motivos para ir a un hotel seguido ;).
    - Listado de hoteles por tipo: Esta bueno tener un listado de hoteles y todos sus datos de contacto, acomodados por cantidad de estrellas y tipo (hotel, hostel, casita en la pradera, etc.) Incluir el precio no es mala idea. Cuando hagas el relevamiento no olvides preguntar a los lugares si hacen algún tipo de plan por grupos, muchos son muy buena onda con esto. Dejales tus datos de contacto y la dirección del sitio web donde vas a poner sus datos, los hoteles tienden a llamarte para avisar si cambian de precios o disponibilidades, esto te ahorra trabajo al momento de mantener actualizada la información, tené en cuenta que van a pasar unos meses entre el relevamiento y el evento.
    - Medio de comunicación para los alojantes locales: Una de las ventajas de estos eventos es conocernos entre nosotros y compartir, tratá de crear un espacio para que los que vienen de afuera y la gente con lugar para alojar gente se contacten.
    - Como llegar desde los hoteles: Agrega algo de la información de transporte a la pagina de los hoteles, al menos básica y de distancia.

- Comida: Suma a todo esto información de diversos tipos de lugares para comer cerca del evento, tené en cuenta las necesidades especiales como por ejemplo celíacos o vegetarianos. Algo de información sobre lugares notables en la ciudad para comer tampoco esta mal, puede servir al que llega antes y quiere conocer un poco la ciudad. Nuevamente te recuerdo, precios.
- Contenidos: Trata de mantener actualizada la información de las charlas y espacios disponibles. No dejes que haya paginas con errores en tu sitio, en vez pone algo que diga porque no esta disponible y cuando estará, tratá de respetar esa fecha, si no tenes certeza date mas tiempo, no prometas imposibles.
- Fechas: Asegurate que el cierre de inscripciones, llamados a charlas, encuestas y todo otro asunto que requiera interacción de terceros tenga fechas límites visibles y repetirlas tanto como sea necesario.

Contabilidad

El dinero siempre es un tema incómodo, sobre todo cuando el evento es un asunto de comunidad y amistad o compañerismo.

Previsibilidad Económica

Para esta área es importante saber algunos datos:

- Elaborá al menos tres presupuestos con diferentes niveles de optimismo, te van a ayudar a priorizar tus gastos y asignaciones de dinero.
- Poné fechas límites para las confirmaciones de sponsors y para el pago de los mismos. Es importante mantener la incertidumbre económica lo mas lejos en el tiempo posible del evento.
- Priorizá en el presupuesto las cosas que prometiste a los sponsors, es un compromiso asumido por toda la comunidad que te apoya y nuestra reputación es importante para nuestra existencia.

Finanzas

Sea cual sea el método que tengas de manejar oficialmente el dinero (nosotros tuvimos ayuda de una Fundación que nos prestó su cuenta y personería jurídica para entregar recibos) tenes que llevar cuentas clarísimas.

Algunos tips:

- Antes de comenzar con todo asegurate de que quien vaya a manejar el dinero, quien venda los sponsorings y vos sepan muy bien cuales son los detalles de facturación para los gastos a realizar, los detalles de los recibos que se darán a quienes entreguen dinero y las salvedades del caso. Algunas básicas son:
  - Que tipo de recibo se entregará al sponsor y que información requerirás del mismo para esto.
  - Que tipo de factura/s podes aceptar y que datos tenes que entregar para esto (te recomiendo que te hagas un papel con estos datos y lo lleves en la billetera desde el primer dia hasta que cierre el evento)
  - Cuanto tardas en emitir recibos.
  - Como se hacen pagos al evento desde el exterior.
- Asegurate que la persona que maneja el dinero tenga toda la información necesaria para poder decidir a donde va la plata
- Hagan una reunión semanal para decidir como destinar presupuestos y saber como viene la cosa.
- Mantengan una planilla de cálculos compartida con los datos de:
  - Sponsors que prometieron compra de paquetes (con información de cuando paga cada uno)
  - Costos de cada cosa a adquirir (con información cuando se ha realizado el pago de la misma y si se tiene la factura)
  - Movimientos de dinero de terceros al fondo de evento (a veces uno tiene que adelantar dinero al evento)
  - Totalizaciones de:
    - Cuanta plata te prometieron
    - Cuanta plata física tenes
    - Cuanta plata debes
    - Cuanto dijiste que vas a gastar
    - Cuanto gastaste.
- La factura de todo lo que gastes NO es opcional y no se entrega dinero si no hay factura. Si alguna organización oficial tiene que rendir cuenta de sus movimientos va a necesitar la factura adecuada para cada gasto. Sé intransigente con esto o

podes causar muchos problemas. Exigí que la factura sea lo mas detallada posible, es tu derecho y te va a simplificar la vida.

- Hace un chequeo de finanzas antes de cada reunión y durante el evento al final de cada día. Es necesario para saber que no hiciste ningún gasto que te olvidaste de anotar (pasa mucho con caja chica cuando estas muy cerca del evento)

Selección de disertantes

Tenemos dos tipos importantes de disertantes:

- Keynotes: Invitados por la organización a hablar, esto es la cereza del postre de las charlas, la parte mas show del evento. Algunos tips a tener en cuenta.
    - Anticipación: La gente que vas a invitar suele ser importante en la comunidad y tiene una agenda apretadita. Tratá de tener los nombres listos y hacer la invitación con la mayor anticipación posible (unos 10 meses esta bien.)
    - Elección popular: Hace una lista de el triple de largo de lo que necesitás y que la gente de la comunidad las ordene por interés. Está demás decir que cuando creás la lista te fijes que te es posible traer a esta gente, por ejemplo de que no tenés que pagarles un combo Luis Miguel a cada uno para que venga. Está bueno que la gente pueda elegir a sus héroes :) y además que te den opciones de backup por las dudas que los primeros no puedan.
    - Perfil del invitado: No siempre se le da la importancia que debería a esto, pero es muy importante. Creo que en general uno disfruta mas cuando el invitado internacional disfruta. Si traes a alguien que va a estar en modo estrella todo el evento suele causar una situación incómoda para todos:
        - Chequeá su participación en la comunidad y comunicación con los demás, suele ser fácil de ver si es alguien social (para los estandares geek de social)
        - El idioma, si traés a alguien de un país que hable otro idioma fíjate que puedas comunicarte lo suficiente, si habla tu idioma es mejor, así puede disfrutar mas el evento. Trata de que hable algo que al ménos un grupito de tus colaboradores hable, o te las vas a ver interesantes
        - Necesidades especiales: Esto es algo a tener en cuenta con todo el mundo, pero como ésta persona es tu responsabilidad durante su estadía, asegurate que puedas satisfacer necesidades especiales de salúd, comida, creencias, etc. (por ejemplo no traigas una persona con problemas respiratorios a una ciudad muy húmeda o un vegetariano a comer un asado)
- Charlistas generales: Si podes hace una encuesta sobre los temas que mas interesan para la elección de charlas, esto seguro te ayudará (o a los que hagan la selección a priorizar). Si una charla es muy interesante y alineada con lo que parece ser el interés de los asistentes considerá la posibilidad de afrontar el gasto de viaje del charlista si el mismo no puede.

Logística pre evento

Mientras te acerca al evento, la carga de trabajo va aumentando, algunos de los siguientes tips ayudan a llegar en tiempo y forma y sin una úlcera.

Marchandaising del evento: Una vez decidido cual va a ser el marchandaising date una semana para averiguar los siguientes datos del mismo:

- Costos: (supongo que vas a elegir la mejor relación precio calidad)
- Tiempos de elaboración: Vas a necesitar esto para saber cuando tenes que encargar cada cosa.
- Lugar de entrega: Esto es importante para saber que tipo de logística de transporte vas a tener que gestionar.
- Packaging: Tenes que saber donde vas a almacenar esto

Con toda esta información y suponiendo que tu calendario financiero lo permite (que debería si comenzaste a organizar el evento con suficiente antelación) procurá encargar todos los items el doble del tiempo de elaboración antes del evento (suponiendo que el item sea algo que dura en el tiempo y que el tiempo no sea ridículamente largo). Esta fórmula no siempre anda así que como regla general no podes tener nada mas tarde de 1 semana antes del evento.

Si te es posible, anda armando los paquetes apenas tengas los elementos, esto te evita corridas de último momento. Si alguien te ofrece llevar material extra exigile los mismos tiempos que te exigis a vos mismo.

NOTA IMPORTANTE: Si va a hacer un souvenir para visitantes o disertantes, asegurate que sea algo que pase la aduana y se pueda llevar en un avión.

Material gráfico: Conseguí imprenta para las cosas que vayas a necesitar y cuando la tengas asegurate que estén cómodos trabajando con formatos varios de archivos y sean abiertos a ayudarte y sugerirte materiales, procesos y demás cosas. Trata de quedarte con la misma gente siempre, acelera los trámites. Si vas a tener que imprimir algo como un libro (en nuestro caso el tutorial de python) conseguite una versión impresa de quien haya sido el maquetador del libro (así tenes algo de que dar ejemplos) y una imprenta con experiencia, la impresión de este tipo de cosas no es trivial.

Badges y material personalizado: Tratá de que el material personalizado sea el mínimo posible, en caso de badges por ejemplo, siempre hay una parte que se puede hacer igual para todos y utilizar stickers o algo similar para completar durante el evento. Si de todas maneras podes tener esto una semana antes es mejor.

Invitados, charlistas y keynotes: Hay algunas cosas a prever para facilitarle la vida a los invitados:

- Mandale a todos estos una carta en su idioma firmada y lo mas oficial y detallada posible, a veces son necesarias en la aduana y en el trabajo.
- Reservate un hotel a quienes son tu responsabilidad, asegurate de mandarles un mail con detalles de que incluye su estadía.

Certificados: Hace todos los certificados necesarios y firmalos antes del evento, es mas fácil destruir certificados no entregados que firmar 200 en un día.

Versión original

La versión original de este artículo puede encontrarse en http://python.org.ar/pyar/HGTTP

Charlas relámpago en PyConAr 2010

Autor: Juanjo Conti

Una charla relámpago es una presentación corta a elección del disertante, no mayor a cinco minutos.

A diferencia de las Charlas Programadas, no hay proceso de aprobación; los oradores simplemente se anotan durante la PyCon. El tema de la charla no tiene porque estar relacionado con Python sino que se puede hablar de cualquier tema.

Este período rápido orienta a los oradores a enfocarse en lo esencial y proporciona a la audiencia unos diez temas durante una hora.

Lo bueno de las charlas relámpago es que en 5 minutos podes enterarte de algo muy copado. Y si el que está hablando es un bodrio, no importa: en 4 minutos se va!

Charlas

Durante la PyCon tuvimos charlas relámpago los dos días de charlas, viernes y sábado. Estas fueron algunas de ellas, con links para seguir leyendo:

- Diego Sarmentero sobre el IDE NINJA: http://www.ninja-ide.org.ar
- Manuel Naranjo sobre sobre cusepy: http://code.google.com/p/cusepy/
- Manuel Naranjo sobre OpenProximity: http://www.openproximity.org/
- Mariano Guerra sobre Python embebido: https://github.com/marianoguerra/talks/raw/master/PyConAr2010/lightning.pdf
- Federico Heinz sobre una campaña en contra del voto electrónico: http://trac.usla.org.ar/e-votrucho
- Roberto Alsina sobre Un web browser transformativo.
- Roberto Alsina: “Una hoja de cálculo en N líneas”.
- Juanjo Conti mostrando un demo de Taint Mode: http://svn.juanjoconti.com.ar/dyntaint/trunk/webdemo/
- Alejandro Cura sobre Deferreds.
- Martín Gaitán sobre la versión 3 de las remeras de PyAr: http://python.org.ar/pyar/RemerasV3/
- Luciano Bello sobre Indexación de documentación grafológica: http://people.debian.org/~luciano/security-doc/
- Naty bidart sobre tests.
- Ricardo Kirkner sobre django-configglue: https://launchpad.net/django-configglue
- Roberto Allende con Como cocinar la wikipedia con dos huevos.
- Emiliano Della Verde Marcozzi sobre SQLAlchemy.
- Hugo Ruscitti mostrando un framework orientado a componentes para hacer videojuegos: http://www.pilas-engine.com.ar/doku.php
- Nueces sobre PyCamp.
- Felipe Lerena sobre Mozilla Argentina: http://www.mozilla-ar.org/
- Tenuki y manuq sobre Karma, un sistema de scoring para la lista de PyAr: http://listas.python.org.ar/listinfo/Karma
- Joaq sobre el Wiki de PyAr.

Fotos de las Keynotes

Autor: Juanjo Conti con fotos del álbum de Machinalis

Keynote

¿Qué es una keynote? En este tipo de eventos una keynote o charla plenaria es el momento donde se para todo. Si bien durante el día hay charlas en distintas aulas, en el momento de las keynotes todo el público es reunido en un mismo salón, por ejemplo un aula magna para escuchar a un disertante de lujo. Por lo general el disertante es algún invitado internacional de relevancia para el tema del evento en cuestión.

En PyConAr 2010 los disertantes keynotes fueron Leah Culver sobre el Zen de Python aplicado a los emprendimientos empresariales y Christiano Anderson sobre MongoDB.

Leah

Culver es una ingeniera en software especializada en desarrollo de aplicaciones web y el framework Django.

Fue co-fundadora y líder de la red social Pownce, la cual fue adquirida por Six Apart en noviembre de 2008 y el sitio web fue apagado.

Es apasionada del open source y las especificaciones abiertas, en particular OAuth y OEmbed. Le gusta probar nuevos lenguajes de programación y se encuentra desarrollando una aplicación para iPhone para Plancast.

Vive en San Franciisco aunque creció cerca de Minneapolis.

Sitio web personal: http://blog.leahculver.com/

Christiano

Anderson es miembro de Python Brasil y desarrollador de Software Libre en Trianguli. Ha sido consultor por diez años, especializado en tecnologías web utilizando especialmente Python y Django.

Ha sido disertante en varios eventos en Brasil y en el exterior.

Su presentación está disponible en: http://www.slideshare.net/canderson/python-and-MongoDB

Sitio web personal: http://christiano.me/

Más fotos del evento

Las fotos de este artículo son de http://www.flickr.com/photos/machinalis/sets/72157625148609499/

El Asado de PyConAr2010

Autor: Juanjo Conti, con palabras robadas de Fisanotti y Batista.

Así se anunciaba originalmente el 8 de septiembre:

El viernes 15 de octubre a las 20hs va a haber una juntada alrededor de un asado al que están invitados todos los disertantes y colaboradores de PyConAr 2010.

En el asado, se trató de poner de manifiesto el lado social del evento, un lugar para parar la pelota, relajarse y charlar.

El asado, o comida final, suele ser después del último día de charlas, pero en esta ocasión, día de la madre y viajes mediante, muchos iban a retornar el mismo sábado a la tarde, por lo que el momento social, de reflexión, algarabía y compañerismo (en ningún orden particular) calló a mitad del evento.

El asado es la escusa

¿Cómo es un asado que no es asado? Originalmente iba a ser asado, pero terminó siendo algo que parece estar muy de moda en Córdoba: "pata". Con el término "pata" se refieren a que uno contrata el servicio de una gente que vienen con una pata graaande de ternera, ya cocida, calentita, mucho pan, algunos tarritos con diferentes salsitas, y la cortan ahí "en vivo", y uno mismo se va haciendo sanguchitos. No me quedó claro si el tomate y la lechuga que había la habían puesto los organizadores, o vienen incluidos en el servicio de "pata". —facundobatista

La visión de los nuevos miembros

Esta bueno ver tanta gente comprometida por la difusión y evolución de una tecnología, al punto de regalar horas de trabajo, viajar desde lejos, compartir lo que conocen e incluso aportar económicamente a la causa, convencidos de que lo que tienen es algo que vale la pena difundir. Viniendo de un pasado no tan lejano de estar metido en otras tecnologías (Microsoft, .Net, etc), todavía me sorprende lo que se logra con una comunidad sin estructuras jerárquicas, títulos oficiales, ni empresas que la banquen.

Y finalmente esta lo humano, el contacto de uno a uno. Lo compartido en los momentos más informales, como el asado o la salida del sábado, es impagable. Conversaciones y debates con gente que además de ser profesionales tienen calidad humana. Gente que no tiene problemas en compartir una solución a un problema, conocimientos, consejos, junto con bromas, buena onda, y una cerveza de por medio :) (aunque yo no tome, igual me parece genial, jaja). —fisadev

Las fotos son cortesía de: http://www.flickr.com/photos/54757453@N00/sets/72157625061739525/

Entendiendo Decoradores en Python

Autor: Juanjo Conti
Juanjo es Ingeniero en Sistemas. Programa en Python desde hace 5 años y lo utiliza para trabajar, investigar y divertirse.
Blog: http://juanjoconti.com.ar
Email: jjconti@gmail.com
Twitter: @jjconti

Una nota sobre lo que sigue

Lo siguiente no es un artículo propiamente dicho, sino la *desgravación* de una charla que fue presentada en forma oral, no desde un medio de audio, sino de mi memoria! Espero que al lector le sea de utilidad.

Introducción

Los siguientes son los tópicos que traté en la charla:

- El principio de todo
- ¿Qué es un decorador?
- Funciones decoradoras
- Decoradores con parámetros
- Clases decoradores
- Decorar clases

El principio de todo

Todo en Python es un objeto

- Identidad
- Tipo
- Valor

En Python todo es un objeto. Los números, strings, listas, tuplas y otras cosas más raras: los módulos son objetos, el código fuente es un objeto. Todo es un objeto. TODO.

Cada objeto tiene 3 características o atributos: identidad, tipo y valor.

Objetos

Veamos algunos ejemplos. El número 1 es un objeto. Usando la función built-in `id` podemos averiguar su identidad. Su tipo es `int` y su valor es obviamente 1.

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

Al ser un objeto, podemos aplicarle algunos de sus métodos. `__add__` es el método que se llama cuando utilizamos el símbolo `+`.

Otros objetos:

```
[1, 2, 3]   # listas
5.2         # flotantes
"hola"      # strings
```

Funciones

Si todo son objetos, las funciones también son objetos.

```
def saludo():
    print "hola"
```

Podemos obtener el id de una función mediante `id`, acceder a sus atributos o incluso hace que otro nombre apunte al mismo objeto función:

```
>>> id(saludo)
3068236156L
>>> saludo.__name__
'saludo'
>>> dice_hola = saludo
>>> dice_hola()
hola
```

Decorador (definición no estricta)

Vamos a tomarnos por un momento una libertad y diremos que un decorador es una *función* **d** que recibe como parámetro otra *función* **a** y retorna una nueva *función* **r**.

- d: función decoradora
- a: función a decorar
- r: función decorada

Podemos aplicar el decorador utilizando una notación funcional:

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

Veamos ahora cómo implementamos un decorador genérico:

Código

```
def d(a):
    def r(*args, **kwargs):
        # comportamiento previo a la ejecución de a
        a(*args, **kwargs)
        # comportamiento posterior a la ejecución de a
    return r
```

Definimos una función **d**, nuestro decorador, y en su cuerpo se define una nueva función **r**, aquella que vamos a retornar. En el cuerpo de **r** ejecuta **a**, la función decorada.

Cambiemos ahora los comentarios por código que haga algo:

Código

```
def d(a):
    def r(*args, **kwargs):
        print "Inicio ejecucion de", a.__name__
        a(*args, **kwargs)
        print "Fin ejecucion de", a.__name__
    return r
```

Cuando ejecutemos una función decorada con el decorador anterior, se mostrará un poco de texto, luego se ejecuta la función decorada y se finaliza con un poco más de texto. Veamos un ejemplo.

En `suma2` nos guardamos la versión decorada de `suma`. Veamos ahora lo que pasa cuando la ejecutamos:

Manipulando funciones

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

```
>>> suma(1,2)
3
>>> suma2 = d(suma)
>>> suma2(1,2)
Inicio ejecucion de suma
3
Fin ejecucion de suma
>>> suma = d(suma)
>>> suma(1, 2)
Inicio ejecucion de suma
3
Fin ejecucion de suma
```

Así mismo podemos guardarnos directamente en `suma` la versión decorada de `suma` y ahora nunca más a lo largo del programa se tendrá acceso a la versión original.

La anterior forma de aplicar un decorador es la forma *funcional*. Tenemos una más linda:

Azúcar sintáctica

A partir de Python 2.4 se incorporó la notación con @ para los decoradores de funciones.

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

suma = d(suma)
```

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

En la porción de código anterior se pueden ver dos ejemplos en donde comparamos las formas de aplicar un decorador.

Lo siguiente es ver ejemplos de decoradores *reales*.

Atención

Anti-ejemplo: el decorador malvado.

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

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

Este decorador es tramposo, por que en lugar de devolvernos una nueva función, nos devuelve un objeto booleano. Obviamente cuando lo intentamos ejecutar, obtenemos un error.

Decoradores encadenados

Su aplicación es similar al concepto matemático de componer funciones.

```
@registrar_uso
@medir_tiempo_ejecucion
def mi_funcion(algunos, argumentos):
    # cuerpo de la funcion
```

Es equivalente a:

```
def mi_funcion(algunos, argumentos):
    # cuerpo de la funcion

mi_funcion = registrar_uso(medir_tiempo_ejecucion(mi_funcion))
```

Decoradores con parámetros

- Permiten tener decoradores más flexibles.
- Ejemplo: un decorador que fuerce el tipo de retorno de una función.

Supongamos que queremos un decorador que convierta a string todas las respuestas de una función. Se usaría de esta forma:

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

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

¿Cómo se implementaría? Veamos una primera aproximación:

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

Esta forma funciona, pero pensemos si podemos hacerlo de una forma más genérica. La siguiente es la forma de utilizar el decorador `typer`:

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

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

```
>>> edad()
25
```

En realidad, `typer` no es un decorador, es una fábrica de decoradores.

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

Notemos que `_typer` es el verdadero decorador, la función externa recibe un parámetro `t` que es utilizado para definir la naturaleza del decorador a crear.

Ahora nos vamos un poco más lejos y veremos:

Clases decoradoras

Características:

- Decoradores con estado.
- Código mejor organizado.

El primer ejemplo es similar a nuestra primera función decoradoradora:

```
class Decorador(object):

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

    def __call__(self, *args, **kwargs):
        # comportamiento previo a la ejecución de a
        self.a(*args, **kwargs)
        # comportamiento posterior a la ejecución de a
```

La siguiente ejemplifica como usarlo:

```
@Decorador
def nueva_funcion(algunos, parametros):
    # cuerpo de la funcion
```

Funcionamiento paso a paso:

- Se instancia un objeto del tipo `Decorador` con `nueva_función` como argumento.
- Cuando llamamos a `nueva_funcion` se ejecuta el método `__call__` del objeto instanciado.

También podemos aplicarlo, utilizando la vieja notación:

```
def nueva_funcion(algunos, parametros):
    # cuerpo de la funcion
nueva_funcion = Decorador(nueva_funcion)
```

Con estos ejemplos vistos, podemos hacer una definición más estricta de decoradores:

Decorador (definición más estricta)

Un decorador es una *callable* **d** que recibe como parámetro un *objeto* **a** y retorna un nuevo objeto **r** (por lo general del mismo tipo que el orginal o con su misma interfaz).

- d: objeto de un tipo que defina el método `__call__`
- a: cualquier objeto
- r: objeto decorado

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

Decorar clases (Python >= 2.6)

A partir de Python 2.6, se permite el uso de la notación con @ antes de la definición de una clase. Esto da lugar al concepto de decoradores de clases. Si bien antes de 2.6 se podía decorar una clase (utilizando la notación funcional), recién con la introducción de este azúcar sintáctica se empezó a hablar más de decoradores de clases.

Un primer ejemplo:

Identidad:

```
def identidad(C):
    return C
```

Retorna la misma clase que estamos decorando.

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

Cambiar totalmente una clase:

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

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

Similar a uno de los ejemplos del principio, el ejemplo nos muestra que lo que retorne un decorador tiene que tener una interfaz Similar a la del objeto que estamos decorando, así tiene sentido cambiar el uso de la versión original del objeto, por una cambiada.

Reemplazar con una nueva clase:

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

```
>>> @reemplazar_con_X
... class MiClase():
...     pass
...
>>> MiClase
<class __main__.X at 0xb78d7cbc>
```

En el caso anterior vemos que la clase bue cambiada completamente por una clase totalmente diferente.

Instancia:

```
def instanciar(C):
    return C()
```

```
>>> @instanciar
... class MiClase():
...     pass
...
>>> MiClase
<__main__.MiClase instance at 0xb7d0db2c>
```

Como último ejemplo de decoradores de clase vemos un decorador que una vez aplicado, instancia la clase y asocia este objeto a su nombre. Puede verse como una forma de implementar el patrón Singleton, estudiado en programación. # cita de singleton a wikipedia

Para terminar:

Dónde encontramos decoradores?

Permisos en Django

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

URL routing en Bottle

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

Standard library

```
classmethod, staticmethod, property
```

Muchas gracias!

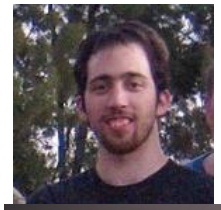
La charla cerraba agradeciendo al público por su atención. Aprovecho en esta ocasión para agradecerles a César Portela y a Juan BC por leer el borrador de esta *desgravación*.

Datos y contacto

PyConAr 2010 - Córdoba - 15/10/2010

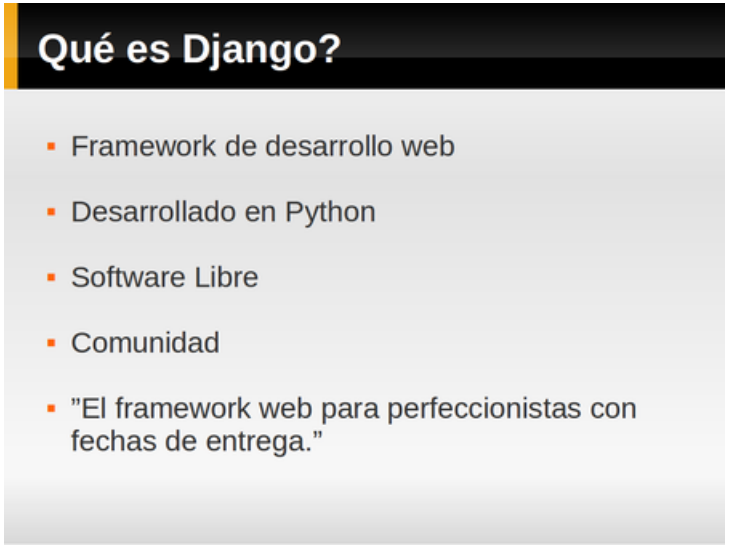
- Comentarios, dudas, sugerencias: jjconti@gmail.com
- Blog: http://www.juanjoconti.com.ar
- Twitter: @jjconti
- http://www.juanjoconti.com.ar/categoria/aprendiendo-python/
- http://www.juanjoconti.com.ar/2008/07/11/decoradores-en-python-i/
- http://www.juanjoconti.com.ar/2009/07/16/decoradores-en-python-ii/
- http://www.juanjoconti.com.ar/2009/12/30/decoradores-en-python-iii/
- http://www.juanjoconti.com.ar/2010/08/07/functools-update_wrapper/
- Slides originales de esta charla: http://www.juanjoconti.com.ar/files/charlas/DecoradoresPyConAr2010.pdf

Introducción a Django



Autor: Juan Pedro Fisanotti
Python es mi lenguaje preferido aunque también me agradan mucho Ruby y Lisp, y Django es el framework de desarrollo web que más me gusta. Linuxero (Ubuntu) desde no hace tantos años, y entusiasta del software libre. Actualmente trabajo con Python principalmente :)

La charla original se llamaba “Mini Introducción a Django”, pero resultó que la versión escrita no es tan “mini”.



Bien, ¿qué es Django? En pocas palabras: es un framework de desarrollo web. Es decir, un conjunto de bibliotecas y herramientas que nos van a permitir crear sitios web. Como ya se deben imaginar, está hecho en Python y por tanto también será Python (con todas sus bondades) el lenguaje que utilicemos para crear nuestros sitios.

Django es software libre, con lo que tenemos acceso a su código fuente para aprender, entender, ayudar a mejorarlo, etc. Y además goza de una comunidad muy grande y activa, lo que ayuda a que se mantenga actualizado, se detecten y corrijan sus errores, tenga documentación actualizada y detallada, y algunas otras ventajas que después vamos a ver (spoiler: muchas aplicaciones útiles ya hechas!).

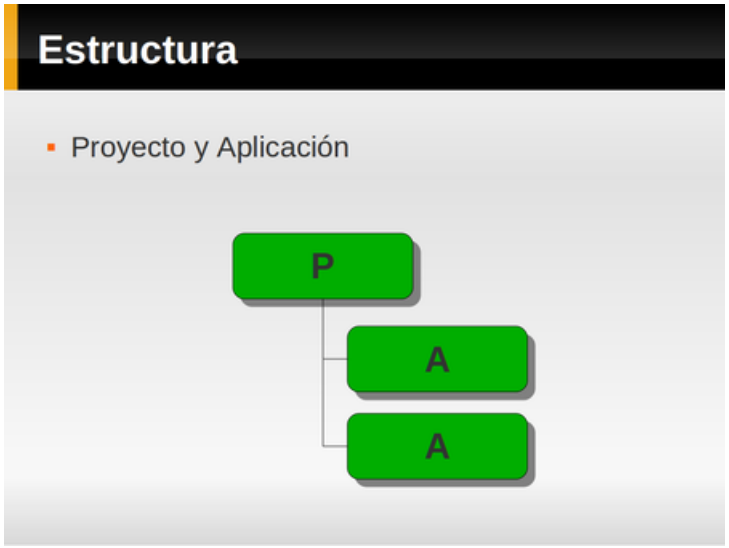
Además, es bueno saber que Django tiene una filosofía muy definida, influenciada por el ambiente donde nació. Los creadores originales de Django trabajaban haciendo sitios para empresas de noticias, donde muchas veces se requerían cambios en cuestiones de días o horas. Y como se trataba de un grupo de desarrolladores “perfeccionistas”, el desafío era llegar a las apretadas fechas de entrega pero escribiendo código de manera correcta, y no haciendo “chanchadas” para que las cosas salieran rápido.

De allí que se dice que Django es “el framework web para perfeccionistas con fechas de entrega”. Django espera facilitarnos la tarea de desarrollo, pero ayudándonos a la vez a escribir buen código.

Y finalmente es un framework que intenta ser flexible, no interponiéndose entre el desarrollador y lo que quiere conseguir. Por ello es muy sencillo reemplazar algunas partes que no nos gustan de Django, con otras que nos gusten o sirvan más.

Hecha la Introducción (y si no se aburrieron y dejaron ya de leer), vamos a conocer a Django.

Lo primero importante a saber es cómo Django nos propone estructurar nuestros sitios:



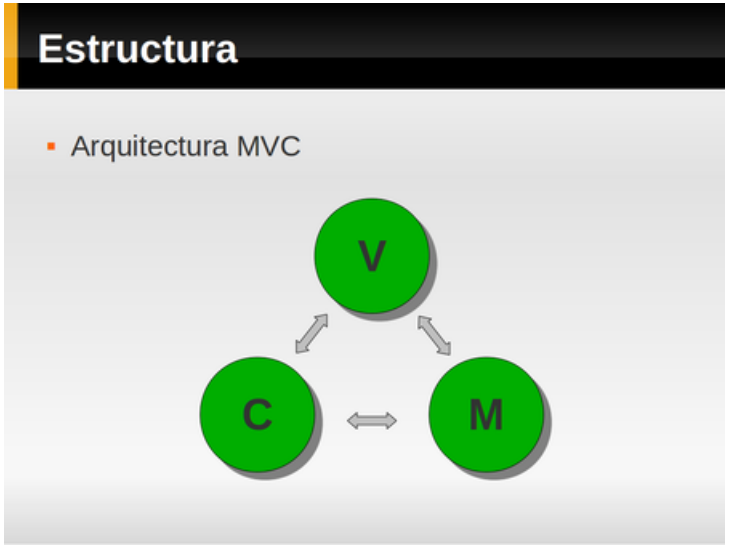
En Django vamos a tener Proyectos y Aplicaciones:

- Proyecto: va a contener la configuración general de nuestro sitio (cosas como la base de datos, email de los admins, etc.), y un conjunto de aplicaciones.
- Aplicaciones: van a ser las que tengan la funcionalidad en sí de nuestro sitio (por ejemplo, la lógica para encontrar la foto del perrito más votado de la semana).

Algo interesante es que Django nos alienta a que las aplicaciones sean lo más desacopladas posibles, de forma de que una misma aplicación pueda reutilizarse en más de un proyecto.

Django mismo ya nos trae muchas aplicaciones útiles para cosas comunes del desarrollo web, como la autenticación de usuarios o la administración del contenido del sitio. Y como Django tiene una comunidad muy grande (y humilde, como verán), también podemos encontrar muchas aplicaciones de terceros para reutilizar y modificar.

Hay aplicaciones que nos ayudan durante el desarrollo (para logging, debug, etc.), aplicaciones para agregar funcionalidad a nuestro sitio (tagging, registración, etc.) y aplicaciones ya armadas que podemos poner directamente en producción (como blogs, CMS, etc.).



Ahora bien, ¿cómo es una aplicación?

Para nuestras aplicaciones Django nos propone seguir la arquitectura MVC ("Modelo-Vista-Controlador"). Para quienes no lo hayan escuchado antes, MVC no es un invento de Django, sino una arquitectura bien difundida que nos propone separar nuestras aplicaciones en tres partes:

- Los Modelos: la parte de nuestra aplicación que define la estructura de la base de datos y se encarga de la comunicación con ella.
- Las Vistas: la interfaz del usuario, con el código que elije qué datos pedirle o mostrarle en cada momento.
- Los Controladores: la parte de la aplicación que elije qué vistas ejecutar en respuesta a las acciones o peticiones del usuario.

Los modelos van a ser clases que representen las cosas que queremos almacenar en la base de datos. Ejemplo: clase Cliente, clase Noticia, etc.

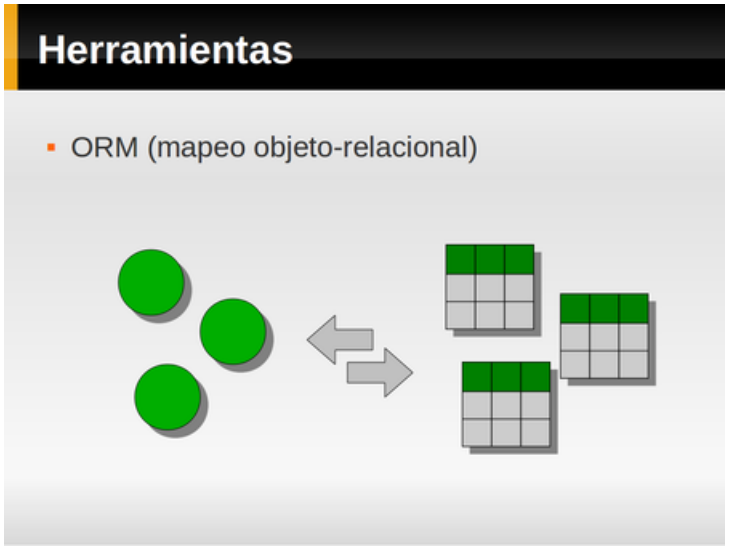
Las vistas van a ser funciones normales de Python, que van a devolver el contenido que debe ser entregado al usuario (página web, imágen, etc.). Ejemplo: la vista "pagina_de_inicio".

Y para los controladores, nosotros sólo vamos a tener que definir qué función (vista) debe ser llamada para cada url. Ejemplo: "cuando el usuario pida la url http://misitio.com/inicio/, ejecutar la vista pagina_de_inicio".

Django se encarga en gran parte de los controladores, y nos provee de herramientas para facilitarnos el desarrollo de las vistas y los modelos.

Vamos a hecharle una mirada a eso...

El paquete de Django trae...



La primer herramienta que nos da Django es un ORM (un “mapeador objeto-relacional”). El ORM va a ser el que nos permita interactuar con la base de datos, y por lo tanto va a ser una de las dos cosas que más vamos a usar comúnmente (así que presten atención, esto es como esos temas que seguro entran en el examen).

Como nosotros programamos con orientación a objetos, lo que vamos a definir y usar son clases. Y como las bases de datos más comunes son relacionales, Django se va a encargar de traducir nuestras operaciones sobre objetos, en sentencias SQL que se van a ejecutar sobre tablas de la base de datos.

Por ejemplo: definimos nuestra clase Usuario, con varias propiedades (nombre, dirección, email, etc.). Luego podemos hacer cosas como crear instancias de Usuario, ingresar valores en sus propiedades, y decirle “che, guardate!”, y Django automágicamente va a armar una sentencia SQL de insert o update según necesite, y la va a ejecutar en la base de datos.

En código, ese ejemplo se vería así:

```
nuevo_usuario = Usuario()
nuevo_usuario.nombre = 'Fisa'
nuevo_usuario.email = 'fisadev@gmail.com'
nuevo_usuario.direccion = 'alguna parte de Argentina'
nuevo_usuario.save()
```

De la misma manera el ORM nos va a permitir leer de la base de datos, filtrar, realizar consultas complejas, etc.

Por ejemplo, si quisiéramos buscar todos los usuarios que tengan 20 años, el código se vería así:

```
veinteanieros = Usuario.objects.filter(edad = 20)
```

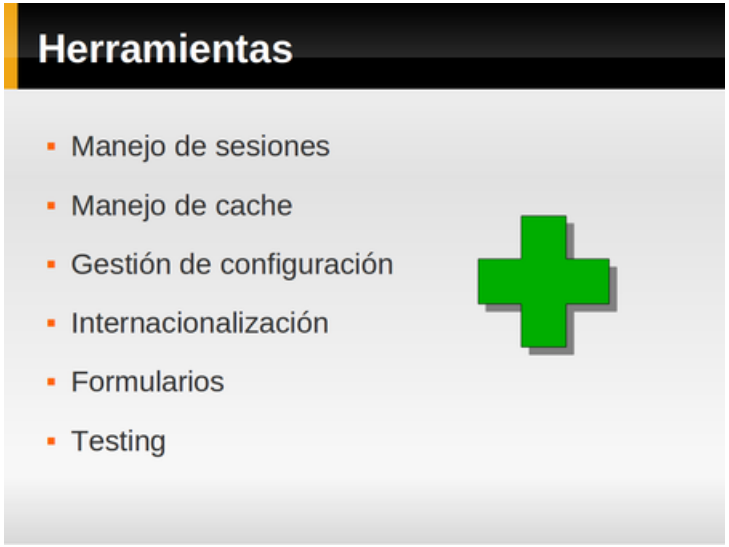
[ya que estamos: la aplicación de autenticación que trae Django ya tiene una clase User que podemos aprovechar :)]

Ya podemos entonces escribir código que lea y guarde cosas en la base de datos, pero eso al usuario no le sirve de mucho. Hay que mostrarle algo. Así que la segunda herramienta que más vamos a utilizar es el sistema de plantillas.

¿Qué es el sistema de plantillas? Es lo que nos va a permitir “injertar” nuestros datos dentro de plantillas html (o xml, o atom, ...) que nosotros definamos.

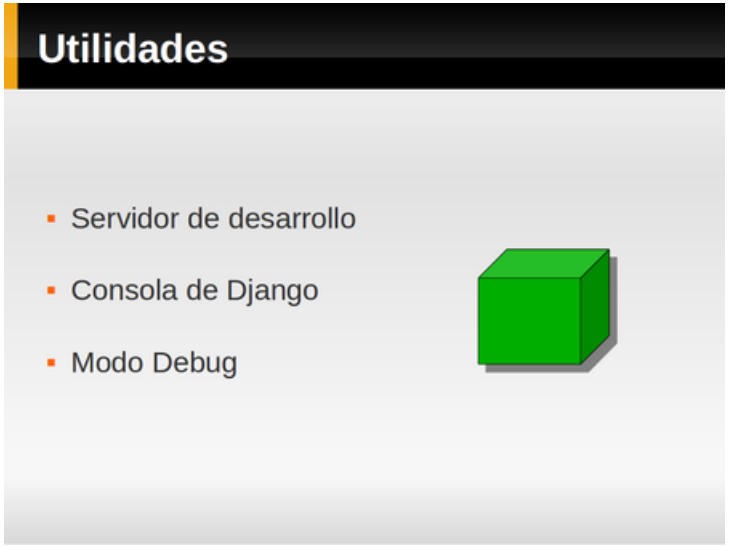
Es decir, vamos a armar nuestro html que esperamos que el usuario vea en su navegador, y dentro del html diremos cosas como “acá tiene que aparecer el nombre del cliente actual”.

El lenguaje de templates (plantillas) además nos permite poner un poco de lógica básica al código html, como ciclos, condiciones, o incluso una especie de herencia de plantillas.



Y hay un número de herramientas extras que también usaremos mucho en nuestros sitios al desarrollar con Django. Siempre es bueno conocerlas para no andar “reinventando la rueda”, ya que además se trata de herramientas que han sido pulidas por las necesidades y uso de mucha gente.

Si algo huele a “esto lo debe hacer mucha gente”, entonces es probable que Django tenga alguna herramienta para facilitar la tarea. Si no la tiene, también es probable que alguien la haya desarrollado como una aplicación. Y si tampoco existe como aplicación, genial! ya descubriste algo con lo cual contribuir :)

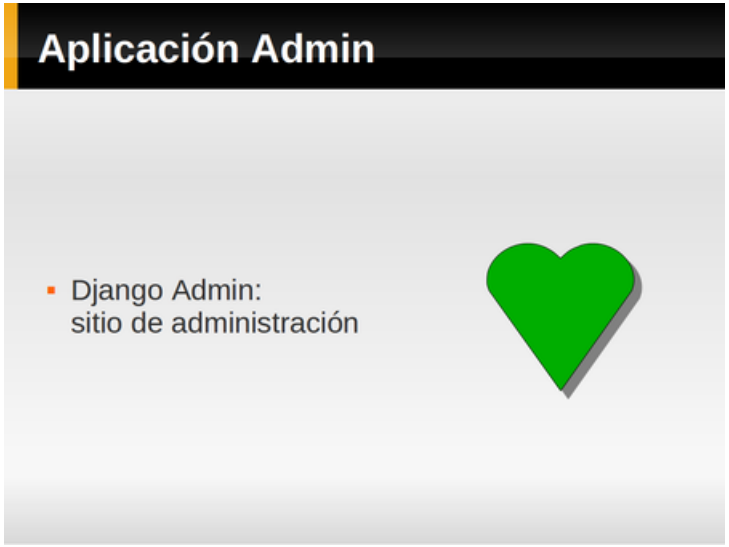


Hasta ahora las herramientas que estuvimos viendo son cosas que utilizaremos más que nada mientras programemos. Cosas que vamos a usar dentro de nuestro código. Pero Django también trae algunas utilidades para facilitar algunas tareas más allá de la codificación.

El servidor de desarrollo nos permitirá ejecutar nuestro sitio web desde el entorno de desarrollo mismo, sin necesidad de configurar un servidor web para ello ni estar haciendo deploys para probar cada cambio que hagamos.

La consola de Django nos permite ejecutar código en una consola interactiva de Python, pero como si fuese nuestro sitio ejecutándose en el servidor. Podemos usarla por ejemplo para interactuar con la base de datos pero con nuestros modelos.

Y el modo de debug, que nos permite ver muchísima información de los errores en la ejecución del código de nuestro sitio. Teniendo el debug activado, frente a un error podemos ver el código que lo generó, las variables que había en memoria en ese momento y sus valores, los parámetros GET y POST recibidos, la configuración actual del sitio, el stack trace de la excepción de Python, etc. Para quienes han trabajado en desarrollo web, saben bien lo que esto vale, ya que es algo que generalmente no poseemos.



Y como postre de esta lista de herramientas y utilidades, tenemos al Admin de Django.

Django Admin es una aplicación que viene incorporada, y que nos ahorra una cantidad impresionante de trabajo. Sí, ya se que suena medio fanático y exagerado, pero es bastante realista lo que dije.

Recordemos que en nuestras aplicaciones podíamos definir modelos. Es decir, clases que representaban las cosas que almacenábamos en la base de datos (clase Usuario, clase Noticia, clase Comentario, etc.).

En los modelos ya definimos todo lo necesario para poder luego operar con esas entidades en la base de datos, crear nuevas, buscar, guardar, modificar, etc. Entonces, ¿qué nos falta para tener una aplicación que nos permita administrar el contenido de la base de datos? Las pantallas (páginas), lógicamente.

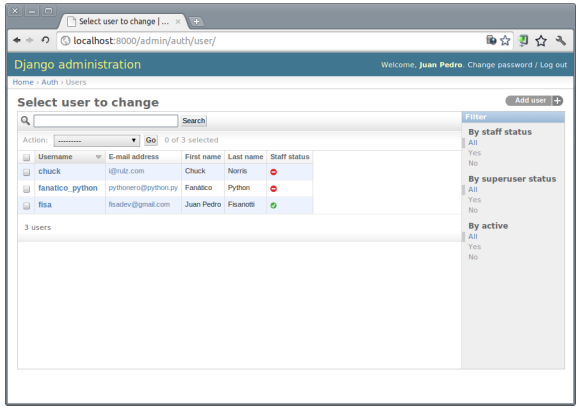
Bueno, el admin de django es precisamente eso. Simplemente le decimos algo como “che, admin! mirá, acá yo dije que tenia clientes en la base de datos, ¿no me mostrás un ABM de clientes?”. Y el admin nos muestra un ABM de clientes. Así de sencillo (si no lo creen, pueden ver el código en el tutorial de Django).

[ABM (o CRUD en inglés) es una aplicación que permita hacer Altas, Bajas y Modificaciones (Create, Read, Update, Delete, en inglés)]

Incluso podemos personalizar la manera en que esos ABM se muestran, con cosas como mostrar filtros por los campos que queramos, decidir que columnas mostrar en las listas, cómo mostrar los campos en la edición, etc.

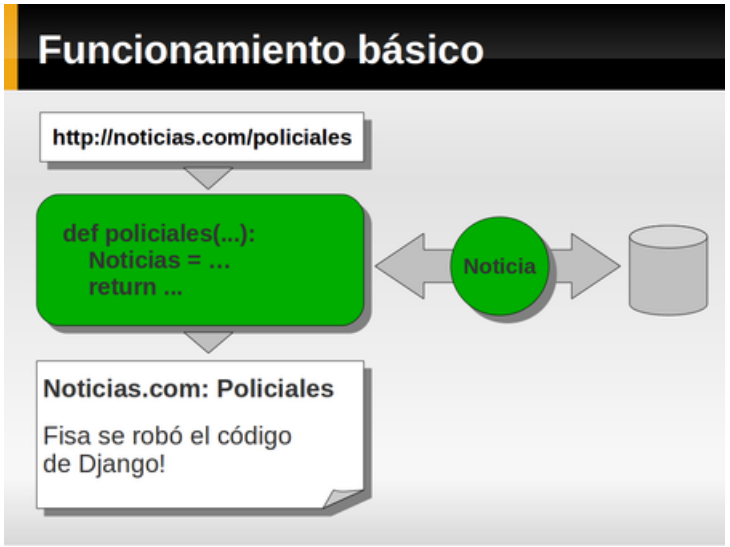
Esta aplicación nos sirve a nosotros desarrolladores durante la creación del sitio, para cargar contenido y ver “cómo iría quedando”. También nos sirve luego en producción, para los administradores o moderadores del sitio. Pero también le puede servir a nuestro usuario final, dependiendo del caso.

Cada vez que veo el admin, me da lástima recordar las horas que dediqué a programar ABMs que no eran ni un cuarto de lo configurables que son en Django :)



Haciendo funcionar las partes

Bueno, ya recorrimos una cantidad interesante de las cosas que Django tiene. La pregunta ahora es: ¿Cómo se engancha y funciona todo esto?



Es más simple de lo que parece:

1. El usuario pide una url (“quiero entrar a http://noticias.com/policiales, y que sea rápido!”)

2. Django se da cuenta de que para responder a esa url, tiene que llamar a una función (una *vista* de ahora en más). Como se imaginan, la llama.

3. En nuestra vista (función “policiales” en el ejemplo), leemos todas las noticias de la base de datos que tengan la categoría policial.

4. Finalmente nuestra vista le dice a Django que le devuelva al usuario un template determinado (en este caso el html de la página de policiales), pero utilizando dentro del template los datos que nosotros le pasamos.

Hasta aquí tenemos una idea global del funcionamiento y la manera de trabajar de Django, pero todo esto puede entenderse mucho mejor si lo vemos con código. Por ese motivo es que les recomiendo ir al tutorial oficial de Django, que pueden encontrar en http://docs.djangoproject.com/en/dev/intro/tutorial01/

En el tutorial pueden ver cómo construir una aplicación sencilla desde cero, y con lo leído en este artículo, seguramente les será más sencillo entender lo que vayan haciendo en su transcurso.

Y si no les es más sencillo, siempre tienen mi mail para quejarse :)

PyAfipWs: facilitando, extendiendo y liberando los Servicios Web de AFIP (Factura Electrónica y otros)

Autor: Mariano Reingart
Analista Programador y Docente. Entusiasta del Software libre y Python, PostgreSQL y Web2Py en particular.
Blog: http://reingart.blogspot.com
Empresa: http://www.sistemasagiles.com.ar

Introducción

PyAfipWs <http://www.pyafipws.com.ar/>_ es una interfase de software libre a los Servicios Web de la AFIP, desarrollado en Python.

Nació por el 2008 de un intercambio de mails en la lista de PyAr!, cuando con el Señor Marcelo Alaniz, consultando sobre unos problemas con Python y webservices, fuimos armando una librería que se adaptara a los requerimientos y ambientes de AFIP (que dicho sea de paso, son bastantes variados y ninguna herramienta en python funcionaba del todo)

La interfase fue “inspirada” en los ejemplos oficiales de la AFIP para PHP, ya que eran los más simples de entender y más cercanos a un lenguaje como Python. Había ejemplos para Java pero estaban incompletos (tienen algunas dificultades con el tema dependencias e incompatibilidades con XML/SOAP) y en .NET (que estaban más completos pero solo para Windows). Igualmente, para nuestro gusto, estos dos ejemplos eran bastante difíciles de seguir, generan código (“artefactos”), etc. etc.

Adicionalmente, de PHP nos inspiramos en algunas bibliotecas de manejo simple de XML y SOAP, que luego dieron lugar a su contraparte en python por parecernos sencillas e intuitivas.

Inicialmente la desarrollamos para los web services de autenticación (firma digital y demás) y factura electrónica, pero con el correr del tiempo fuimos agregando otros servicios de AFIP, como bienes de capital - bono fiscal electrónico, facturas de exportación, código de trazabilidad de granos, depositario fiel y los nuevos servicios para mercado interno.

El proyecto ha madurado y algunas librerías que desarrollamos o adaptamos han sido liberadas separadamente:

- PySimpleSoap (http://pysimplesoap.googlecode.com/): para manejo simple y completo de webservices
- PyFPDF (http://pyfpdf.googlecode.com/): para generación de PDF de manera más fácil y fluida.

Ambas han sido integradas a web2py (http://www.web2py.com) para tener un marco de trabajo completo para desarrollar aplicaciones web de gestión.

Factura electrónica en Python:

A modo de ejemplo, vamos a mostrar un como autorizar una factura electrónica en pocas líneas.

Primero se debe solicitar un ticket de acceso (utilizando un certificado y clave privada tramitadas previamente), que nos permitirá posteriormente utilizar el servicio web de factura electrónica:

```
from pyafip.ws import wsaa

# crear ticket acceso, firmarlo y llamar al ws
tra = wsaa.create_tra()
cms = wsaa.sign_tra(tra, "homo.crt", "homo.key")
ta_xml = call_wsaa(cms, wsaa.WSAAURL)
```

Detrás de escena se usa M2Crypto (que es el enlace de OpenSSL para encriptación en Python), necesario para firmar la solicitud de acceso en XML.

Del ticket de acceso extraemos el Token y Sign (usando SimpleXMLElement para analizar y convertir el XML en una estructura de objetos y datos python), que contienen los códigos de seguridad requeridos en los otros webservices:

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

Una vez obtenida la autenticación para acceder a los servicios web, vamos a proceder a solicitar autorización para emitir una factura electrónica.

Para ello creamos una conexión con el servidor y llamamos remotamente al procedimiento de autorización:

```
from pyafip.ws import wsfe

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

# autorizar factura electronica (obtener 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']
```

Si todo funciona bien, esto nos devolverá el CAE (Código de Autorización Electrónico), necesario para confeccionar la factura electrónica.

Aquí termina el tema de los webservices en python, para la generación de la factura (por ej. en PDF), envió por correo y demás, ver PyRece a continuación.

Lenguajes legados

Más allá de las aplicaciones en Python, esta biblioteca es compatible con lenguajes como Visual Basic, ASP, Fox Pro, Cobol, Delphi, Genexus, PowerBuilder, PHP, .Net, Java, ABAP (SAP), etc. y cualquier lenguaje/aplicación que pueda crear objetos COM (automatización) (http://es.wikipedia.org/wiki/Component_Object_Model) en Windows (por ej. Excel o Access).

Esto se logra fácilmente utilizando PythonCOM (parte de las extensiones win32), envolviendo una clase común de python para que pueda ser expuesta a otras aplicaciones, definiendo los métodos y atributos públicos, el nombre expuesto y demás, por ej:

```
class WSAA:
    "Interfase para el WebService de Autenticación y Autorización"
    _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}"
```

Una vez registrado la interfaz, se la puede llamar desde cualquier otra aplicación con esta tecnología, por ej, en Visual Basic sería:

```
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  ' setear token y sign de wsaa
WSFE.Sign = WSAA.Sign
WSFE.Cuit = "3000000000" ' CUIT del emisor

ok = WSFE.Conectar(url)

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

En nuestro caso fue muy útil y posibilitó a muchas aplicaciones contemplar estas nuevas funcionalidades (webservices, encriptación, etc.) con modificaciones menores, que de otro modo hubieran sido muy difíciles o imposibles.

Archivos de Texto y Línea de Comandos

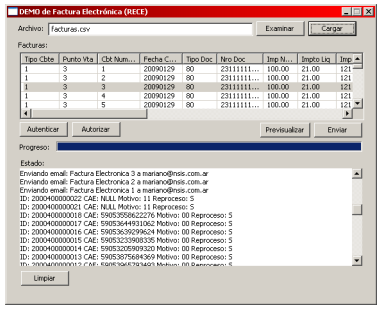
Si bien la interfaz COM es muy útil para aplicaciones relativamente modernas, todavía hay lenguajes o entornos de muy difícil acceso, donde prácticamente la única forma de interoperabilidad son los archivos de texto.

Viendo que lenguajes como Cobol manejan archivos con campos de longitud fija (y esto ya era soportado por el aplicativo RECE de SIAP), tomamos ese camino, que con Python fue bastante directo.

La interfaz incluye herramientas como RECE.PY y RECEX.PY que por línea de comandos reciben y procesan los archivos de entrada, guardando los resultados en archivos de salida. Esto es controlado con un archivo de configuración (RECE.INI) que utiliza define las URL, certificados y rutas a utilizar.

PyRece: ¿SIAP libre?

La historia no termina aquí, ya que viendo algunas dificultades del Aplicativo RECE o los servicios en linea para hacer facturas electrónicas (funcionalidad limitada, demoras o complejidades, etc., sobre todo para los contribuyentes que no poseen sistema de gestión), desarrollamos un aplicativo visual (de “escritorio”) para facilitar y extender las posibilidades brindadas por AFIP:



La interfaz del usuario es una pantalla simple pero contempla:

- Lectura de planilla de calculo CSV (en vez de archivo de ancho fijo)
- Autenticación y Autorización on-line en el momento
- Generación del PDF personalizable (textos, logos, lineas, etc.) con la imagen de la factura
- Envió por mail del PDF a los clientes (con un breve mensaje configurable)

Todas estas funciones no están disponibles (en su conjunto) en el aplicativo o sitio web de AFIP (algunas pueden realizarse con limitaciones, pero no de una forma totalmente ágil y transparente).

Esta aplicación a su vez demuestra que PyRece es una posibilidad concreta de desarrollarlo con Python un Sistema Integrado de APlicaciones (SIAP) de software libre, en este caso como alternativa al aplicativo RECE de factura electrónica, pero hay otros casos donde se podría hacer lo mismo si los servicios web estuvieran disponibles (por ej., IVA para DD.JJ., SICOSS para seguridad social., etc.).

FE.py

Por último, hemos desarrollado una herramienta que unifica los distintos webservices (factura electrónica nacional, exportación, bienes de capital, etc.) e integra todo lo expuesto anteriormente, utilizando una base de datos para almacenar e intercambiar la información, generando las facturas en PDF, pudiéndolas cargar desde y hacia archivos de texto, envío por email o FTP, entre otras cuestiones.

En general utiliza PostgreSQL, pero también se puede usar otras bases de datos (PyODBC o SqLite).

El Proyecto

Cerrando el artículo, incluimos algunos comentarios sobre como se desarrollo el proyecto.

Un tema resuelto fue el modelo de negocios, sobre todo conociendo que hay otras alternativas cerradas y decidimos mantenernos en el camino del software libre, encontrar la combinación justa para poder competir no fue un tema menor.

Para los personas que no conocen Python y desean evaluar la interfaz, ponemos a disposición un instalador para demostración totalmente funcional en homologación (testing). Ofrecemos un soporte comercial pago para los que necesiten realizar consultas, soliciten corrección ajustes y deseen tener acceso al instalador para producción y actualizaciones futuras que liberemos.

Todo el código fuente esta publicado en el repositorio de GoogleCode (http://pyafipws.googlecode.com) bajo la licencia GPL3, con los respectivos scripts e instructivos de instalación.

El resultado creemos que ha sido bastante satisfactorio, posibilitando extender y mantener el proyecto financieramente, contribuyendo al software libre y a la comunidad con herramientas alternativas y superadoras, y empezando a crear una grupo de desarrolladores interesados en el tema.

La interfaz tiene miles de descargadas desde el sitio del proyecto, y muchas empresas y compañías importantes nos han contratado el soporte comercial.

Esto no ha sido totalmente sin contratiempos ni esfuerzos, por lo que para finalizar van algunos comentarios y recomendaciones:

- Los instaladores y paquetes fueron fundamentales para que las personas puedan evaluar los productos, sobre todo en tecnologías no tan difundidas como Python, y principalmente para Windows, que ha sido el mayor mercado para esta interfaz.

- La documentación y ejemplos fueron otro punto importante, y por experiencia es un tema donde se debe profundizar constantemente, aún en los casos que parezca que es suficiente (incluimos preguntas frecuentes, recortes de código, aclaraciones importantes, etc.).
- Los cursos de capacitación y talleres son muy productivos (hemos realizado 2 en en la ACP, a quienes agradecemos), permiten extenderse un poco más sobre el tema y conocer a los interesados.
- Para la difusión nos han ayudado mucho blogs, noticias, eventos de software libre, etc. A medida que el proyecto fue apareciendo en los buscadores fue creciendo su popularidad y utilidad real para los usuarios.
- El soporte comunitario en nuestro caso no ha sido efectivo, la lista de correo y los sistemas de tickets/issues no se usaron mucho, tampoco ha habido muchas contribuciones y revisiones al código fuente, quizás por el carácter sensible y/o particular del tema (amén que la filosofía del software libre todavía no ha sido muy bien transmitida en algunos ambientes).

Esperamos que este artículo haya servido de una visión general sobre el tema, cualquier información adicional la pueden encontrar en las siguientes direcciones:

- Sitio del proyecto: www.pyafipws.com.ar (http://www.pyafipws.com.ar)
- Grupo de Noticias: groups.google.com.ar/group/pyafipws (http://groups.google.com.ar/group/pyafipws)
- Soporte comercial y documentación: www.sistemasagiles.com.ar/trac/wiki/PyAfipWs (http://www.sistemasagiles.com.ar/trac/wiki/PyAfipWs)
- Minisitio AFIP factura electrónica: www.afip.gov.ar/fe (http://revista.python.org.ar/1/html/www.afip.gov.ar/fe)

InfoPython - Midiendo el Valor de la Información de Mass Media con Python.

Autor: *Juan Bautista Cabral*
JBC conoció python una solitaria noche del 2007. Desarrolló su proyecto de grado de la carrera Ingeniería en Sistemas con este lenguaje utilizando el framework Django y trabajó 1 año desarrollando evaluadores de información usando nuestro querido reptil.
blog: http://jbcabral.wordpress.com
mail: jbc.develop@gmail.com

Infopython es una librería para la valoración de medios de información que utiliza teorías formales provenientes de las ciencias sociales. Inicialmente, eran un conjunto de módulos dispersos que utilizaba en mi trabajo; luego de unos días de refactoring y paciencia logré unificarlos en una única librería.

Trasfondo

Existen diferentes teorías sociológicas para determinar la importancia de los medios sobre la opinión pública, las mismas analizan la información que estos emiten desde el punto de vista del emisor, el receptor o ambos.

Por citar un ejemplo, la **Teoría de Información de Shannon** es una formalización matemática de una de una teoría sociológica conocida como la **Aguja Hipodérmica**.

En el caso de **Infopython** la teoría que nos ocupa es conocida como **Agenda-Setting** (en el futuro se planean agregar otras); la cual postula que los medios de comunicación de masas tienen una gran influencia sobre el público al determinar qué historias poseen interés informativo, y cuánto espacio e importancia se les da. El punto central de esta teoría es asignar una prioridad para obtener mayor **audiencia**, mayor **impacto** y una determinada **conciencia** sobre la noticia.

Los Medios

En **Infopython**, antes de mencionar **como** son los procesos de cálculo del valor de la información es necesario determinar **qué** es lo que vamos a medir.

Así definimos informalmente que para nuestro dominio un medio de información es:

Un elemento emisor sobre el cual quiero efectuar una medición del valor de su información. Esta información tiene como características: ser homogénea, dar una "sensación" de unidad, y ser medible.

Siendo:

- **Homogénea:** Toda información que emite el medio tiene que tener características comunes. Sino dada su extrema variación interna el medio se nos hace imposible de medir.

  Por Ejemplo:

  > un canal de televisión, para nuestro caso, no califica como "medio" ya que cada programa y franja horaria tiene **muy** diferentes niveles de audiencia y contenido; en este caso es mejor tomar como unidad cada programa de televisión como el medio a medir.

- **Sensación de Unidad:** Es más fácil entender este concepto por medio de un ejemplo:

Si yo digo que mi medio de información son "Revistas de Deporte" da una sensación de que este elemento no es "un solo medio", pero sin embargo cuando cambio la definición del medio, a "La revista de deportes Goles y Faroles" esta sensación sí esta presente.

- **Medible:** Si de un medio que definimos no podemos extraer datos cuantitativos, no tiene sentido para nosotros.

Formalizando

Llegado el punto en el cual ya tenemos definidos **que es un medio** para nuestro dominio, podemos definir "matemáticamente" un modelo que se ajuste a la definición previa de la "Agenda-Setting"; así proponemos lo siguiente:

El Valor de la información de un medio es una función de la audiencia y el impacto, descartando la conciencia ya que es difícil o imposible de medir

O lo que es lo mismo:

VALOR = F(AUDIENCIA, IMPACTO)

Siendo:

- **VALOR:** Es la importancia del medio dada la teoría.
- **AUDIENCIA:** A cuanta gente le llega la información del medio.
- **IMPACTO:** Qué tanta importancia le da la audiencia al medio.

Ahora como ultimo se propone como **F** a:

F(AUDIENCIA, IMPACTO) = AUDIENCIA * IMPACTO

Se elige la función '*' (Multiplicación) por los siguientes motivos:

- **Refleja mejor la variación de los valores:** si un parámetro crece o decrece mucho, hace variar mucho al valor.

Supongamos el siguiente caso:

Un medio el cual es seguido por un número bajo de audiencia **10** pero tiene un alto impacto **1.000**. Esto puede darse si esas pocas personas pertenecen a un grupo de influencia (asesores presidenciales por ejemplo).

En este caso el valor de la información sería **10.000**, y dado qué valor definimos como "mucho" y qué valor definimos como "poco", este valor es "grande" (mucho). Esto, lo podemos considerar como correcto, ya que cualquier medio que pueda influir en personas importantes debería tener un alto valor.

Manteniendo los valores, pero cambiando nuestra función por **AUDIENCIA + IMPACTO** el valor de la información sería **1.010** el cual, manteniendo el razonamiento anterior sigue siendo alto.

Ahora, si reemplazamos el valor de la audiencia por un número grande **1000**, y mantenemos el del impacto; el valor de la función original sería **1 millón** y en el segundo caso **2.000**.

Si pensamos que ahora impactamos probablemente sobre 1000 asesores presidenciales el valor **2.000** se queda chico, ya que estamos en presencia de un medio que probablemente genere un impacto a nivel global. Lo cual evidencia que la multiplicación representa mucho mejor la variación de parámetros.

- **El valor se anula ante la ausencia de audiencia o de impacto**: Cuando la audiencia o el impacto son **0 (cero)** (Nadie ve o nadie presta atención al medio) el valor de la información también es **0 (cero)**.

Esto no es trivial ya que nos sugiere que la información no vale nada si nadie le interesa verla o nadie le presta atención.

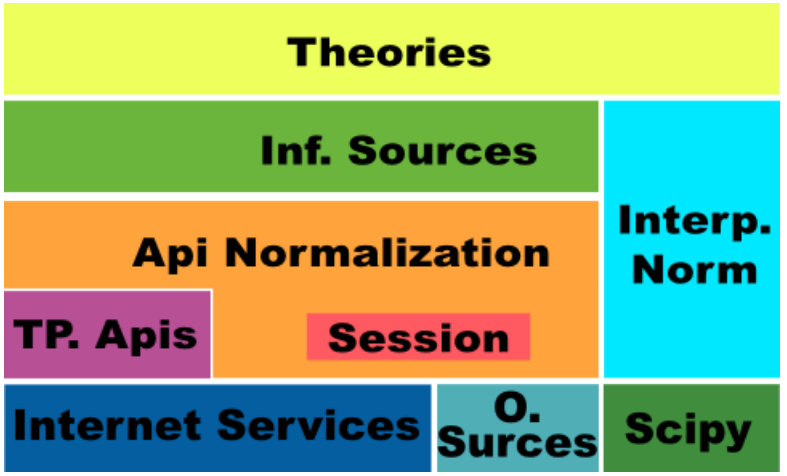
Infopython

Dado que existe una amplia variedad de servicios públicos que extraen estadísticas y datos sobre nuevos medios (web, twitter, etc), como por ejemplo:

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

Por citar algunos. **Infopython** se centra en brindar un API sencilla para valorar a través de agenda-setting (en el futuro se implementarán otras teorías) a los medios independientemente de su tipo, utilizando los servicios antes mencionados

Arquitectura:

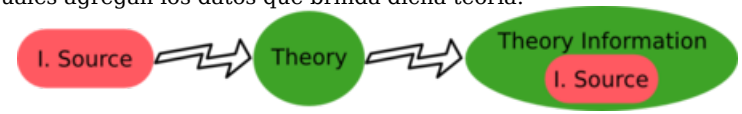


Analizamos Cada Capa:

- **Internet Service:** Corresponde a los distintos servicios que existen en la web para la extracción de estadísticas y datos de los nuevos medios.
- **Other Sources:** Son otros datos que con los que se alimenta a **Infopython**, como ser Bases de datos, plantillas excel, etc.
- **Scipy:** Es una biblioteca de código abierto de algoritmos y herramientas matemáticas.

  Esta se encarga del procesamiento numérico necesario.
- **Third Parties Apis:** Son librerías de terceros que se conectan a servicios que existen en la red. Por ejemplo:
  - tweepy que sirve para manipular datos de twitter.
  - koutpy que se conecta a Klout
- **Session:** Esta sub-capa es un módulo que se encarga de centralizar todas las configuraciones necesarias para acceder a los servicios de internet.
- **Interpolation Normalization:** Esta es una capa de abstracción para los diferentes interpoladores que posee Scipy y define algunos nuevos, todos con la misma API.

- **API Normalization:** Se encarga de convertir todas las respuestas de todos los servicios de internet y las API's de terceros a estructuras comunes (diccionarios) utilizando de ser necesarios los datos que posee la session.
- **Information Sources:** Son las clases que representan nuestras fuentes de información. Las mismas están conectadas de manera "auto-mágica" a las diferentes API's Normalizadas.
- **Theories:** Esta capa posee módulos que definen el comportamiento y los cálculos de las teorías implementadas en la **Infopython** (para la versión actual solo Agenda-Setting). Cada teoría encapsula los medios de de información en "nodos" los cuales agregan los datos que brinda dicha teoría.



Ahora definida toda la teoría, y toda la arquitectura, podemos mencionar cómo se trabaja con la librería:

1. **Configurar la sesión:** Consiste en brindarle a la capa de sesión todas las api key (mecanismos de autentificación de servicios de tercero) que requiera.

Ejemplo:

```
from infopython import session

# Listado de todas las llaves OBLIGATORIAS de la librería
session.NEEDED_KEYS

# configura la session con las llaves v0, v1, ...
session.set(v0=1, v1=2...)

# retorna el valor de una llave
session.get("v0")

# borra la session
session.clear()
```

En la versión actual todas las NEEDED_KEY son obligatorias y la sesión es inmutable.

2. **Crear los medios: Crear los medios de información sobre los cuales** se desea consultar su valor. En esta versión de **Infopython** se brinda clases para **2** medios:

- `WebPages:` Representa una página web independientemente si ésta es un perfil de twitter o un blog, o lo que fuera. Se sugiere como mecanismo de medición de audiencia los servicios de Compete (http://www.compete.com/) o los de Alexa (http://www.alexa.com/).

  Y como mecanismo de medición de impacto Page Rank (http://es.wikipedia.org/wiki/PageRank), ya que si Google dice que la importancia de información es ésta, no vamos a discutir con Google.

  Ejemplo del Api de `WebPage`:

```
from infopython.isources import webpages

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

google.id # devolveria "google.com"
google.url # devolveria "http://google.com"
google.html # El contenido en HTML de "http://google.com"
google.text # El texto del HTML de "http://google.com"

google.get_info("compete") # la informacion de compete de
                           # "google.com" utilizando el
                           # key de compete suministrado
                           # en la session
```

- `TwitterUser:` Representa un usuario de Twitter y NO sus tweets

  Se sugiere como mecanismo de medición de audiencia la cantidad de followers; y de impacto la información suministrada por Klout (http://klout.com/)

  Ejemplo del Api de `TwitterUser`:

```
from infopython.isources import twitteruser

yo = twitteruser.TwitterUser("leliel12")
yo.id # leliel12
yo.username # leliel12
yo.get_info("tweepy") # la informacion de tweepy del usuario
```

```
                    # "leliel12" utilizando el key de
                    # Twitter suministrado en la session
```

3. **Crear Evaluadores:** Consiste en crear **callables** (funciones o métodos) que reciban un medio de información como parámetro y devuelvan los valores que se asumirán como audiencia o impacto. Por ejemplo si decidimos que nuestra isource `WebPage` extraerá su **audiencia** de **Compete** y su **Impacto** de **Pagerank**, la funciones deberían ser similares a estas:

```
# extrae los unique visitors de compete de la WebPage que recibe como
# parámetro
aud = lambda w: w.get_info("compete")["metrics"]["uv_count"]

# Extrae el valor de page rank de la WebPage que recibe como parámetro
imp = lambda w: w.get_info("pagerank")["pagerank"]

Si a la agenda no le sumistramos alguno de los evaluadores, ésta tratará
de  usar los interpoladores suministrados.
```

4. **Crear los interpoladores:** Los interpoladores se utilizan como segunda alternativa a la extracción de **audiencia** e **impacto**, por lo que cada agenda recibe 2 interpoladores: un interpolador de audiencia y uno de impacto.

   Así el interpolador de **impacto** recibirá como valor para interpolar **“X”** a la **audiencia** y devolverá un valor **“Y”** correspondiente al **impacto***.

   Ahora, si lo que deseamos es interpolar el valor de la **Audiencia**, el interpolador recibirá como valor **“X”** el **Impacto** y devolverá un valor **“Y”** correspondiente a la **Audiencia**.

   Se mostrará un ejemplo en conjunto más adelante.

5. **Crear la/s agenda/s:** Al crear las agendas se les debe suministrar diferentes datos:

   - Qué tipo de medio de información medirá.
   - Una lista de medios de información a medir(opcional).
   - Un extractor de datos de audiencia (opcional).
   - Un extractor de datos de impacto (opcional).
   - Un interpolador de audiencia (opcional).
   - Un interpolador de impacto (opcional).

   Se mostrará un ejemplo en conjunto más adelante.

6. **Evaluar los nodos:** La agenda posee métodos para ordenar los `ISources` según su valor, para luego ser iterada y así generar un ranking de importancia de cada medio.

   Al iterar sobre la `Agenda`, ésta devuelve varios `ASNode` los cuales son estructuras de datos que encapsulan a los medios y agregan atributos correspondientes a **Audiencia**, **Impacto** y **Valor** así como también fecha y hora de cuando fue creado el nodo.

Ejemplo Completo

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

# Configuramos la session.
# Todas estas llaves son de fantasía y para una prueba real cualquier
# Usuario puede registrarlas en la pagina de cada aplicación.
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")

# Creamos dos webpages
google = webpages.WebPage("google.com")
yahoo = webpages.WebPage("yahoo.com")

# Saca cosos
aud = lambda w: w.get_info("compete")["metrics"]["uv_count"] # audiencia
imp = lambda w: w.get_info("pagerank")["pagerank"] # impacto

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

# Creamos la agenda
# Esta agenda tratara de extraer los valores de audiencia e impacto con su
# 'valuators',  en caso de volver 'None' lo intentará con sus interpoladores.
# Si estos vuelven a devolver None, se retornará como valor 0.0 y se calculará
# el valor del medio con ellos.
ag = agenda.AgendaSetting(itype=webpages.WebPage,
                          inf_sources=[google, yahoo],
                          audience_valuator=aud,
                          impact_valuator=imp,
                          impact_interpolator=itp,
                          audience_interpolator=itp)
```

```
ag.rank() # ordenamos la agenda por el valor de cada medio

# Iteramos sobre cada ASNode e imprimimos los valores de audiencia e impacto.
for i in ag:
    print i.id, "%s + %s = %s" % (i.audience, i.impact, i.value)
```

Más Métodos de la Agenda

Suponiendo que tenemos una instancia, la misma agenda del ejemplo anterior `ag` y el `WebPage`, `google`:

```
ag.value_of(google) # devuelve el valor de google (audiencia + impacto)
ag.impact_of(google) # devuelve el valor del impacto de google
                     # o sea dado lo que definimos como evaluador de
                     # impacto haría la llamada:
                     # return google.get_info("pagerank")["pagerank"]

ag.audience_of(google) # devuelve el el valor de la audiencia de google
                       # osea dado lo que definimos como evaluador de audiencia
                       # haría la llamada:
                       # return google.get_info("compete")["metrics"]["uv_count"]

ag.wrap(google) # Devolvería un ASNode con los valores de audiencia,
                # impacto y valor de la información de google

ag.count(google) # Devuelve cuantas veces aparece este medio en la agenda

ag.remove(google) # elimina la primer ocurrencia google en la agenda

ag.append(google) # agrega google a la agenda

ag.for_type # Devolveria para que tipo de isource fue creada esta agenda
            # WebPage para nuestro ejemplo

ag.audience_valuator # None o la función de calculo de audiencia

ag.impact_valuator # None o la función de calculo de impacto

ag.audience_interpolator # None o el interpolador de audiencia

ag.impact_interpolator # None o el interpolador de impacto
```

Comparando 2 Agendas

En el módulo `agenda` existe una función que es muy útil para evaluar varias agendas con diferentes medios de información.

Esta función retorna una lista de `ASNode` ordenada de ambas agendas.

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

# 2 agendas con diferentes tipos de medios.
ag1 = agenda.AgendaSetting(isource=webpages.WebPage)
ag2 = agenda.AgendaSetting(isource=twitteruser.TwitterUser)

# itera sobre todos los medios de informacion de ambas agendas
# ordenados por 'value'.
for i in agenda.rank_isources(ag1, ag2):
    print i
```

Nota Final: Test

Al bajar la librería (http://bitbucket.org/leliel12/infopython/) lo primero que debe hacerse es correr el test con los siguientes pasos:

1. **Correr**

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

2. Configurar `test.cfg` con las llaves de las API's correspondientes.

3. **Correr ahora si**

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

Conclusión

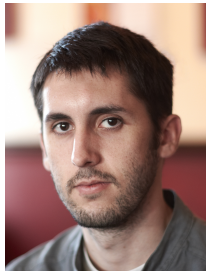
Como vimos **Infopython** provee una manera uniforme para la valoración de la información. En versiones futuras se planea introducir otros tipos de mass-media ya que por ejemplo,**IMDB** y **GoogleBooks** provee información vía API's de medios tradicionales (películas y libros); o, yendo mas allá, **LinkedIn** información bastante confiable de perfiles laborales.

También es posible la integración con el procesamiento de lenguaje natural con NLTK o alguna herramienta de la web semántica.

Enlaces:

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

Como generar archivos .exe e instaladores para una aplicación python



Autor: Mariano Guerra Cordobés, 25 años, Ingeniero en Sistemas recibido en la UTN Córdoba, Programador Python y miembro de Pyar hace demasiado tiempo ;). Creador de emesene.

Este documento describe los pasos necesarios para crear un archivo ejecutable de una aplicación python y como generar un instalador y una versión portable para dicha instalación.

Este documento asume que la aplicación se basa en GTK pero debería funcionar con menores cambios en otros toolkits.

Porqué un instalador

- se requiere instalar muchos componentes a mano por el usuario final para una sola aplicación
- muchos instaladores pequeños
- difíciles de encontrar
- difícil encontrar las versiones exactas que funcionan en conjunto
- requiere instalarlos en un orden definido
- rezar
- algunas veces incluso haciendo todo bien puede no funcionar
- fácil de automatizar y documentar para replicar con cada nueva versión
- liberar al usuario final de los problemas para poder usar la aplicación

Componentes requeridos

- python
- todas las librerías utilizadas por la aplicación
- py2exe
- nsis
- tiempo y suerte

Instaladores

Aquí se listan los links a los instaladores de todos los componentes usados en el ejemplo.

- 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

Orden de instalación

Algunos instaladores son independientes de otros, pero para evitar posibles problemas recomiendo la instalación en el siguiente orden.

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

Tareas extra

- setear la variable de entorno PATH para agregar el path a la instalación de python
- probar la instalación con una pequeña aplicación gtk

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

Prueba con una aplicación de ejemplo

Cree un repositorio con una aplicación de ejemplo para probar los pasos, la aplicación esta disponible en github acá:

http://github.com/marianoguerra/PyGtkOnWindows

Pasos

- descargarla
- descomprimirla
- ejecutar python setup.py py2exe
- copiar los directorios lib y share de la instalación del runtime de gtk (no de la instalación de pygtk) al directorio dist
- copiar todos los archivos del directorio dll al directorio dist
- borrar los locales y temas no usados de los directorios copiados a dist (yo solo dejo el theme MS-Windows)
- crear la siguiente estructura de directorios dentro de dist: etc/gtk-2.0
- dentro de ese directorio crear un archivo llamado gtkrc con una linea como la siguiente dentro:
    - gtk-theme-name = “MS-Windows”
    - podes cambiar el tema usado manteniendo otro theme dentro de share/themes y cambiando el nombre del theme en gtkrc
- right click en ejemplo.nsi y seleccionar “Compile NSIS Script”
- right click en ejemplo-portable.nsi y seleccionar “Compile NSIS Script”
- deberías tener el instalador y la versión portable disponibles
- para probar que funciona correctamente, correr el instalador y la versión portable en una instalación de windows sin los paquetes que instalaste anteriormente

Probar con una aplicación real

Ahora para sentirlo mas real, creemos un instalador y una versión portable de un programa real, en este caso, un proyecto personal llamado emesene 2 (http://www.emesene.org/).

Pasos

- descargarlo de http://github.com/emesene/emesene
- descomprimirlo
- copiar setup.py and ez_setup.py al directorio emesene
- cd emesene
- correr python setup.py py2exe
- cd ..
- copiar los directorios lib y share de la instalación del runtime de gtk (no de la instalación de pygtk) al directorio dist
- copiar todos los archivos del directorio dll al directorio dist
- borrar los locales y temas no usados de los directorios copiados a dist (yo solo dejo el theme MS-Windows)
- crear la siguiente estructura de directorios dentro de dist: etc/gtk-2.0
- dentro de ese directorio crear un archivo llamado gtkrc con una linea como la siguiente dentro:
    - gtk-theme-name = “MS-Windows”
    - podes cambiar el tema usado manteniendo otro theme dentro de share/themes y cambiando el nombre del theme en gtkrc
- right click en emesene.nsi y seleccionar “Compile NSIS Script”

- right click en emesene-portable.nsi y seleccionar “Compile NSIS Script”
- deberías tener el instalador y la versión portable disponibles
- para probar que funciona correctamente, correr el instalador y la versión portable en una instalación de windows sin los paquetes que instalaste anteriormente

Notas

- obtengo algunos de los dlls requeridos de portable python (http://www.portablepython.com/) e inkscape (http://inkscape.org/)

Depuración y defragmentación de memoria

Author: Claudio Freire

Introducción

Esta charla se planificó para darse en 20 minutos. Si no la pueden leer en 20 minutos, releer y releer hasta que tome 20 minutos :-p

Bueno, en realidad el texto aquí presente está sólo ligeramente basado en la charla real. Soy de improvisar mucho en las charlas, sólo tengo a las diapositivas como guía general de cómo va a progresar la charla, y este formato no se adapta tan bien a la improvisación. Además que ni en pedo me acuerdo de las palabras exactas que usé ;)

Pero en serio… es un tema fumado, releer y releer hasta entender.

Cuando empecé a buscar tema para la charla, me decidí por la fragmentación de memoria porque antes yo creía que era un mito. Ok, sí, puede suceder… pero ¿a alguien le sucede? ¿Es un problema? Qué ingenuo de mi parte fue dudar de que ambas respuestas fuesen sí.

En la PyConAr 2010 (y aquí en este espacio ahora) no sólo tuve la oportunidad de transmitir alrededor de 4 meses de revolver las entrañas de Python tratando de averiguar qué sucedía en mis aplicaciones (donde sufrí de la fragmentación de memoria), sino que aprendí yo mismo que este problema es mucho más común de lo que esperaba.

Cuando comenzó la charla, pregunté: ¿alguien sabe lo que es la fragmentación de memoria?. Exactamente cero personas levantaron la mano. Exactamente esa cantidad esperaba yo. Pero al final, varios se me acercaron diciendo que lo mismo que me pasaba a mí (y yo describía, y describiré abajo) les estaba sucediendo.

Según mi estimación post-PyCon, un 10% de los presentes (asumo desarrolladores Python) sufren el problema, y a un 3% de ellos les trae problemas operativos serios. Así que el tema fue mucho más relevante de lo que yo esperaba.

¿Cómo administra memoria Python?

Antes de ver qué es esto de la fragmentación, debemos estudiar algunos detalles internos de la administración de memoria en Python.

¿Cómo dicen ustedes que administra la memoria Python? ¿Malloc? *(una función en C que reserva memoria para quienes no saben C)*

Pues no. Python tiene requerimientos inusuales y muy específicos, si utilizara malloc para satisfacer todas sus necesidades de memoria sería muy ineficiente. Python utiliza, en cambio, una serie de técnicas y estrategias diseñadas para minimizar las llamadas a malloc, llamadas que son muy lentas para usar en el núcleo de la máquina virtual de Python.

- Pools
  - Enteros
  - Flotantes
- Listas libres
  - Tuplas
  - Listas
  - Frames *(sí, los frames son objetos y hay que administrarlos también)*
- Arenas

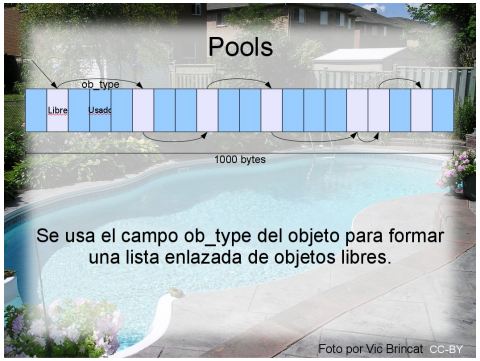
Veamos de qué se tratan uno por uno

Pools

No son piscinas.

Son arreglos, vectores de objetos del mismo tipo, que Python utiliza comúnmente para acelerar la creación y destrucción de objetos muy comunes, como lo son los enteros y los números de coma flotante.

Cuando Python necesita un objeto de estos no crea uno, crea muchos (tantos como entren en un bloque del pool), y mantiene una lista enlazada de qué objetos tiene libres dentro de cada bloque:



Estructura de un pool de objetos

En un pool, el campo **ob_type** (presente en todos los PyObject) se utiliza para enlazar los objetos libres. Cuando se necesita un objeto nuevo, se restaura el campo ob_type (trivial), y en general ya no hace falta más inicialización, por lo que es muy rápido.

En un pool de objetos, la creación y destrucción es muy rápida, y según el tipo de objeto, se puede ahorrar un montón de inicialización (en el caso de enteros y números de coma flotante esto es muy cierto), los objetos permanecen bien empaquetados en la memoria, todos juntitos, y en general todo funciona muy bien.

Listas libres

La idea de no tener que pedir memoria para crear o destruir objetos que son muy comúnmente creados y destruidos es algo que puede generalizarse desde los pools, a cualquier tipo de objeto (no sólo a objetos de un tipo particular), incluso objetos de tamaño variable (donde tenerlos todos empaquetados en un arreglo o pool no es factible).

Cuando se hace esto, se tiene listas libres:

Estructura de una lista libres

Las listas libres son muy parecidas a un pool, pero no se mantiene un arreglo de objetos, sino simplemente la lista enlazada de objetos libres. Cuando se destruye un objeto, se puede decidir ubicarlo en la lista libre en vez de efectivamente liberar la memoria, para poder aprovecharlo y reutilizarlo más tarde.

Esta idea de listas libres ahorra muchas llamadas a malloc, y es particularmente útil para cadenas, listas y tuplas, que son objetos de tamaño variable muy intensamente utilizados en Python y donde un pool no sería una idea práctica. También son la razón principal por la cual Python es particularmente sensible a la fragmentación de memoria, ya vamos a ver por qué.

Nótese que también los frames utilizan listas libres. Los frames son objetos, también de tamaño variable (pues necesitan una pila, espacio temporal para las variables y objetos temporales que nuestro código genere), también muy intensamente utilizados en Python (se “crea” uno cada vez que se hace una llamada a una función). Las listas libres de frames son una optimización muy importante (ahorran mucho tiempo de creación puesto que los frames son costosos de crear), pero también contribuyen a la fragmentación de memoria (como toda lista libre).

Una cosa importante a recordar de las listas libres en Python es que la decisión de si destruir un objeto o ubicarlo en la lista libre se hace al momento de dereferenciarlo (cuando su conteo de referencias llega a cero). Una vez ahí, ahí permanece hasta que sea reutilizado. Python, para tomar esta decisión, tiene una serie de límites - X frames, Y tuplas de tamaño 1, Z tuplas de tamaño 2, W cadenas, etc... (el límite suele ser 100 en cada caso)

Arenas

Ok... y el resto de los objetos. ¿Usan malloc?

Sí y no.

Usan arenas. Que no es de donde sale el vidrio, sino una mezcla entre pools, listas libres y malloc.

Para objetos pequeños, Python mantiene una *lista de pools* por cada tamaño concreto (recordemos que los pools necesitan objetos del mismo tamaño pues son vectores). Cada pool tiene su lista libre, y cada pool tieke 4Kb de tamaño. Esto toma el nombre de arena.

Para objetos grandes (más de 256 bytes), Python llama a malloc directamente.

Como los tamaños de objetos de Python crecen de a 8 bytes (por su estructura), entonces hay exactamente 32 arenas.

Todos los objetos de Python se crean con este mecanismo de arenas, incluso los que usan listas libres.

Las arenas también introducen un problema de fragmentación interna, puesto que ningún bloque de la arena puede ser liberado hasta que todos los objetos que viven en él sean liberados.

Fragmentación

Ahora, veamos lo que es la fragmentación de memoria:

Mapeo de una memoria fragmentada

> Si negro es espacio usado, y blanco es espacio libre, se puede ver aquí como hay mucho espacio libre pero es inusable para objetos más allá de un determinado tamaño, por no ser espacio libre contiguo

Puesto simple, la fragmentación de memoria se produce cuando hay mucho espacio libre, pero no es contiguo. Como en el mapa de memoria que se ve arriba, hay mucho espacio libre, pero es inusable para objetos grandes, puesto que, a diferencia de un archivo que puede ser dividido en fragmentos en el disco, la memoria de un objeto necesita ser contigua.

Así que, a diferencia de la fragmentación en un sistema de archivos, la fragmentación de memoria hace inusable a la memoria. Si quisiera crear un objeto grande, digamos de unos megabytes, debería utilizar el espacio que está hacia el final del mapa (o sea *extender la imagen virtual del proceso*). Esto efectivamente hace malloc cuando se encuentra con esta situación.

El efecto inmediatamente visible es un uso ineficiente de la memoria disponible. Si mi programa necesita 2GB de memoria en teoría, podría estar pidiéndole 4GB al sistema operativo (porque tiene muchos pedacitos reservados que no puede utilizar). Si tengo mucha mala suerte, esto podría hacer que mi sistema swapee. Si tengo más mala suerte, thrashea, y se muere.

Veamos un ejemplo de código que fragmenta la memoria:

```
>>> 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
>>>
```

Luego de esto, top nos dice:

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

O sea, aunque según los cálculos el programa tenía que consumir 800M de memoria, efectivamente consume 1.6G. El doble.

¿Por qué es esto?

Bueno, porque el ejemplo lo pensé específicamente para que cree un 50% de huecos inutilizables. La memoria está fragmentada, pues, en un 50%.

Pero hay algo más grave. Si hago:

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

Obtengo de top:

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

Si repito el ejemplo de fragmentación, puedo comprobar que esos 1.5G están efectivamente libres para python:

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

Pero si intento liberarlos (para el sistema operativo), no puedo.

¿WTF?

Enter Guppy

Guppy es un pecesito rojo comunmente encontrado en las peceras de todos lados. Esos pecesitos chiquitos, esos se llaman guppy.

Posta.

También es una biblioteca de extensión para Python que contiene un módulo, heapy, que me permite hacer diagnóstico de la memoria.

Posta.

Guppy

Veamos un ejemplo de cómo usarlo:

```
>>> 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
```

O sea, Python (por el simple hecho de levantarlo) ya consume 1.7MB. En objetos python. Heapy no cuenta lo que no son objetos Python, así que lo que reporte heapy es todo memoria utilizada directamente por objetos Python.

Esto son cadenas, listas, diccionarios, arrays, pero **no** objetos mmap o memoria utilizada por bibliotecas de extensión (ej: superficies SDL en pygame).

Diagnostiquemos ahora entonces:

```
>>> 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
```

Ok, como habíamos calculado, más o menos 800M (850M) en objetos Python. Eso dice heapy.

```
>>> 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 nos dice que Python ocupa de nuevo 1.7MB. Top sigue diciendo 1.6G. Yo le creo a top.

Sucede que de hecho, el resto es espacio "libre" (libre para Python, no para el sistema operativo)

Haciendo un análisis diferencial, conseguiremos algo de perspectiva en el asunto:

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

Resulta en 3 snapshots del heap. *heap1*, como está al iniciar Python. *heap2*, luego del experimento, y *heap3* luego de "liberar" todo, y dos "diferenciales", *cosas_nuevas*, lo que hay en heap2 de nuevo (que no está en heap1), y *basura*, lo que hay en *heap3* que no está en *heap1* (o sea, lo que no se liberó).

```
>>> cosas_nuevas
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
```

Cabe preguntar: ¿Sólo 850M de cadenas? ¿Y los otros 800M para completar los 1.6G?

Bueno, sucede que la memoria se parece a un queso gruyere en este momento. Hay 800M en cadenas relativamente pequeñas, pero como en cada paso yo liberaba la mitad de ellas (`ll = ll[::2]`), también tengo 800M de espacio libre inutilizable. Porque en cada paso, también, necesito cadenas un poquito más grandes, y no se puede reutilizar los huecos.

A ver qué pasa al de referenciar todo:

```
>>> basura
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
```

¡Ahá!

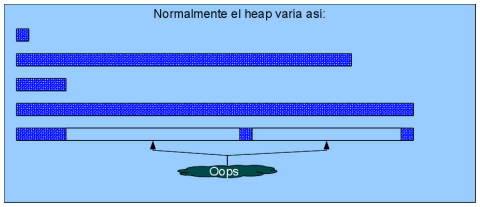
¡Esto es importante!

Esos 29 objetos evitan que se pueda achicar el heap. Lo que me lleva…

Montón (heap)

…al heap.

Normalmente el heap se agranda y se achica.



Ciclo de vida del montón

> El montón se expande y contrae, pero en cada ciclo puede quedar “basura”, o capaz objetos útiles vivos, que impiden que se contraiga del todo. La memoria que queda en el medio no puede ser utilizada por otros procesos, sólo está libre para Python.

Como se ve en la figura, cada vez que se achica, no lo hace completamente. A veces quedan objetos vivos en direcciones elevadas - como el montón no puede fragmentarse (no se puede liberar un espacio del medio del montón, sólo puede agrandarse o achicarse), esos objetos mantienen la memoria del medio reservada para Python. Python puede reusarla, pero el resto del sistema operativo no.

Eso daña el caché de disco, daña otros procesos (capaz otros procesos Python, en un webserver puede suceder que tengamos más de un worker corriendo python), daña la performance general del sistema.

Adivinen quiénes tienen la costumbre de dejar objetos vivos en altas direcciones de memoria…

…así es. Las listas libres. Acá, con guppy encontramos 29 objetos, probablemente todos que están vivos gracias a alguna lista libre que los mantiene vivos. Vemos que un par de ellos son Frames, como decía antes, los Frames causan este tipo de problemas.

Todos queremos saber cómo evitar estos problemas, así que:

Guppy tips

- No dejar basura por el piso
- Si se van a crear muchos objetos pequeños, crear los *persistentes* primero, y los *transientes* al final.
    - Compilar código (ej: usar eval o hacer imports) genera cadenas permanentes, llamadas *cadenas internadas*, así que compilar on-demand también es algo a evitar.
    - SQLAlchemy y muchas otras bibliotecas tienen cachés internos, investigar y estar al tanto de estas políticas.
- Siempre que sea posible, preferir pocos objetos grandes a muchos objetos pequeños:
    - Listas de strings → strings separados por comas. O pipes. O enter. O lo que sea.
    - Listas de números → `array.array` o `numpy.array`
- Barrer de vez en cuando
    - Si se mantienen caches con expiración, limpiar el caché regularmente para quitar elementos expirados
    - A veces se puede “desfragmentar” la memoria, reconstruyendo estructuras persistentes como los cachés

      De hecho, el garbage collector de java hace esto automáticamente, y muchos proyectos buscan implementar un garbage collector similar para Python, pero la API de extensión de Python, la Python/C, lo hace difícil al permitir punteros directos a los PyObject, estructuras que representan los objetos en Python)*.
- El cambio es bueno
    - No crear estructuras eternas.
    - Los caches siempre expiran.
    - Los threads se renuevan.
- La casa es para vivirla, la oficina es para trabajar
    - Siempre que sea posible, realizar tareas intensivas en memoria en un subproceso, que al terminar libera la memoria y deja todo limpito y ordenado.
    - El subproceso es la oficina, ahí se trabaja.
    - El proceso padre es mi casa, ahí se vive.

Links útiles

- **Los slides**: http://python.org.ar/pyar/Charlas#Depuraci.2BAPM-n_y_defragmentaci.2BAPM-n_de_memoria_en_Python
  - **Cómo mapear memoria**: http://python.org.ar/pyar/MapeandoMemoria
  - **Heapy**: http://guppy-pe.sourceforge.net/

NINJA-IDE, Un IDE Pensado para Python

Autores: Diego Sermentero, Santiago Moreno
Estudiantes de Ing. en Sistemas que les gusta desarrollar software libre por amor al arte.
Página: http://ninja-ide.org
Twitter:

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

Como comenzó NINJA-IDE?

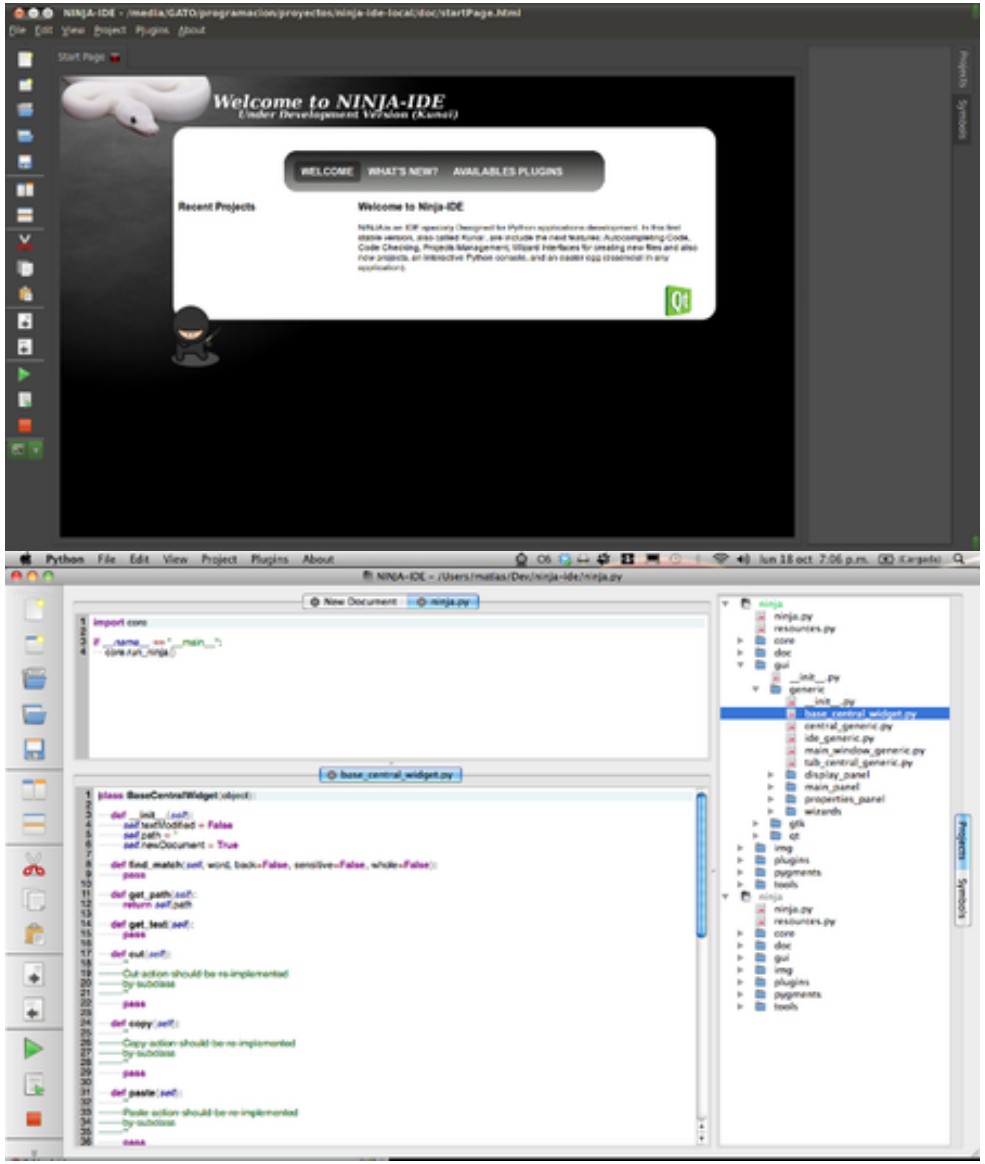
NINJA-IDE nació por unos mails enviados a PyAr, cuya temática suele escucharse con bastante frecuencia: **“Qué buen IDE para Python puedo usar?”**, **“Por qué no hay un IDE para Python que tenga tal o cual característica?”**, y las respuestas a estos mails siempre terminan siendo mas o menos las mismas, ya que los IDEs actuales que encontramos disponibles, en su gran mayoría, no estaban diseñados para Python, sino que brindaban la opción de incorporarlo mediante algún Plugin y de esta forma se solía estar utilizando IDEs muy pesados diseñados para otros fines, donde el soporte para Python en realidad era mínimo, y aquellos que si eran para Python terminaban siendo muy orientados a un Framework especifico o no eran Libres. Entonces, motivados por el desafío que representaba, y por ideas interesantes que se plantearon en la lista de correo, decidimos encarar este proyecto enfocándonos en “qué características debería tener un buen IDE para un programador Python”.

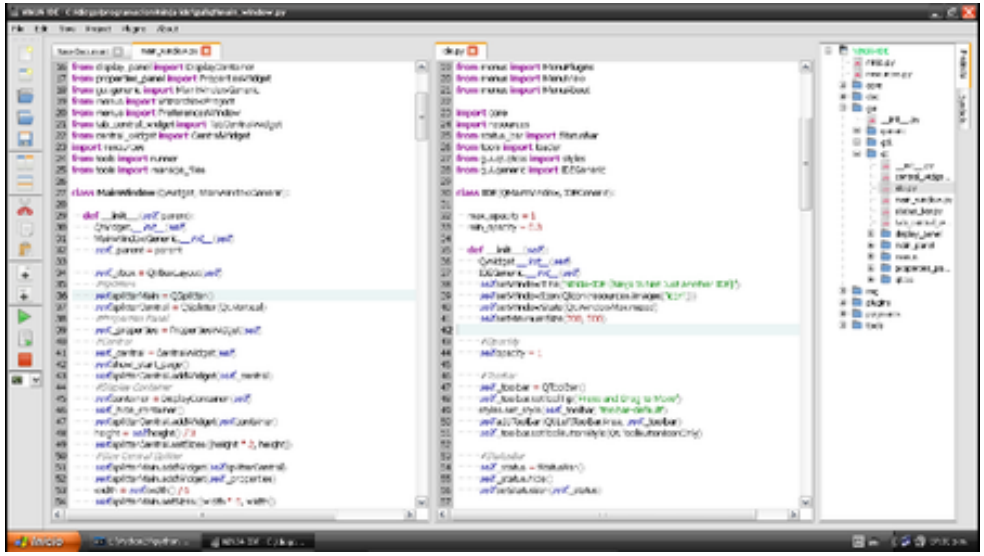
Con esto en mente comenzamos el desarrolló de NINJA-IDE, el cual posee su nombre por el acrónimo recursivo: “Ninja Is Not Just Another IDE”. El IDE tiene apenas unos 7 meses de desarrollo, pero gracias a las ganas y las horas de programación que le dedicamos, ya podemos contar con una versión 1.1 del mismo, con muchas funcionalidades implementadas, y desde su primer semana de desarrollo NINJA-IDE es utilizado para desarrollar NINJA-IDE, lo cual a su vez nos ayuda para encontrar bugs y mejorar la usabilidad y prácticidad de la aplicación, a través de la experiencia y uso continuo de la misma.

El proyecto esta desarrollado bajo licencia libre GPL3 y puede conseguirse el código a través de: http://ninja-ide.googlecode.com

Algunas de las features actuales del IDE son:

- Funcionalidades típicas de cualquier IDE para el manejo de archivos, de Tabs, indentación automática, Zoom en Editor, etc.
- Al estar escrito en Python y utilizar PyQt, es multiplataforma y fue probado en sistemas Linux, MAC OS X y Windows.
- Resaltado de Sintaxis para un gran variedad de lenguajes (si bien esta centrado en Python, brinda el resaltado de sintaxis para otros lenguajes más para comodidad del programador).
- Posibilidad de usar una Consola Python desde el mismo IDE.
- Permite el manejo de Proyectos en el IDE, reconociendo los mismos como Proyectos Python y a través del IDE crear nuevos archivos y carpetas, borrar archivos existentes, creación automática de archivos “__init__” con la información dentro de ese módulo, etc.
- Permite ocultar y reubicar todos los paneles de la interfaz de una forma muy simple, permitiendo que sea adaptado a los gustos del usuario.
- Permite ver más de un Editor al mismo tiempo de forma vertical u horizontal.
- Extensible a través de la incorporación de Plugins (los cuales pueden crearse utilizando un Plugin de NINJA-IDE para mayor simplicidad).
- Maneja sesiones del IDE, para recordar que archivos y proyectos se encontraban abiertos cuando se cerró y los recupera al abrir nuevamente una instancia del mismo.
- Soporte para Auto-completado (siendo un auto-completado especifico del objeto al que se esta accediendo).
- Notificación de actualizaciones.
- Y muchas características más!





Quienes desarrollamos NINJA-IDE?

NINJA-IDE comenzó siendo desarrollado por Santiago Moreno y Diego Sarmentero, y a la semana de haber comenzado el proyecto ya estaba siendo utilizado para desarrollar el mismo. Gracias a la gente de la Lista de PyAr, Blogs, etc. en muy poco tiempo la difusión del proyecto hizo que pudiéramos estar contando con Reporte de Bugs por parte de Usuarios, Sugerencias en la lista de correo de NINJA y hasta con aportes de código por parte de usuarios y colaboradores, de los cuales algunos pasaron a formar parte de NINJA-IDE con el rol de commiters, como es el caso de: Martín Alderete y Matías Herranz. Y otros con el rol de contributors, como: Juan Carlos Ojeda y Cesar Vargas.

Esta fuerte colaboración y participación que estamos recibiendo de la comunidad permite que NINJA-IDE pueda crecer cada día más, mejorando e implementando características que los usuarios necesitan. A su vez los comentarios que recibimos de personas usando actualmente NINJA-IDE, nos motivan a seguir trabajando duro en esta herramienta, con la cual deseamos simplificar aún más el desarrollo de aplicaciones Python.

Para ponerse en contacto con cualquiera de los miembros del proyecto se puede utilizar:

- La Lista de Correo de NINJA-IDE: http://groups.google.com/group/ninja-ide
- La Lista de Correo de NINJA-IDE (Development): http://groups.google.com/group/ninja-ide-dev

- Canal IRC de NINJA-IDE: http://ninja-ide.org/support.html
- Reporte de Bugs: http://code.google.com/p/ninja-ide/issues/list
- NINJA-IDE en Twitter: http://twitter.com/ninja_ide

Actualmente nos pone muy contentos como desarrolladores de NINJA-IDE poder haber conseguido el Tercer Puesto como “Mejor Software Libre” y el Primer Puesto como “Software Libre que necesita más donaciones” (Premio otorgado al proyecto con mayor espectativa de crecimiento), dentro del marco del concurso de ***`Portal Programas <http://www.portalprogramas.com/software-libre/premios/software-libre-donaciones>`_** habiendo conseguido estas menciones teniendo el proyecto solo unos meses de existencia. Y a su vez, formar actualmente parte del Showcase de Aplicaciones del programa Qt Ambassador.

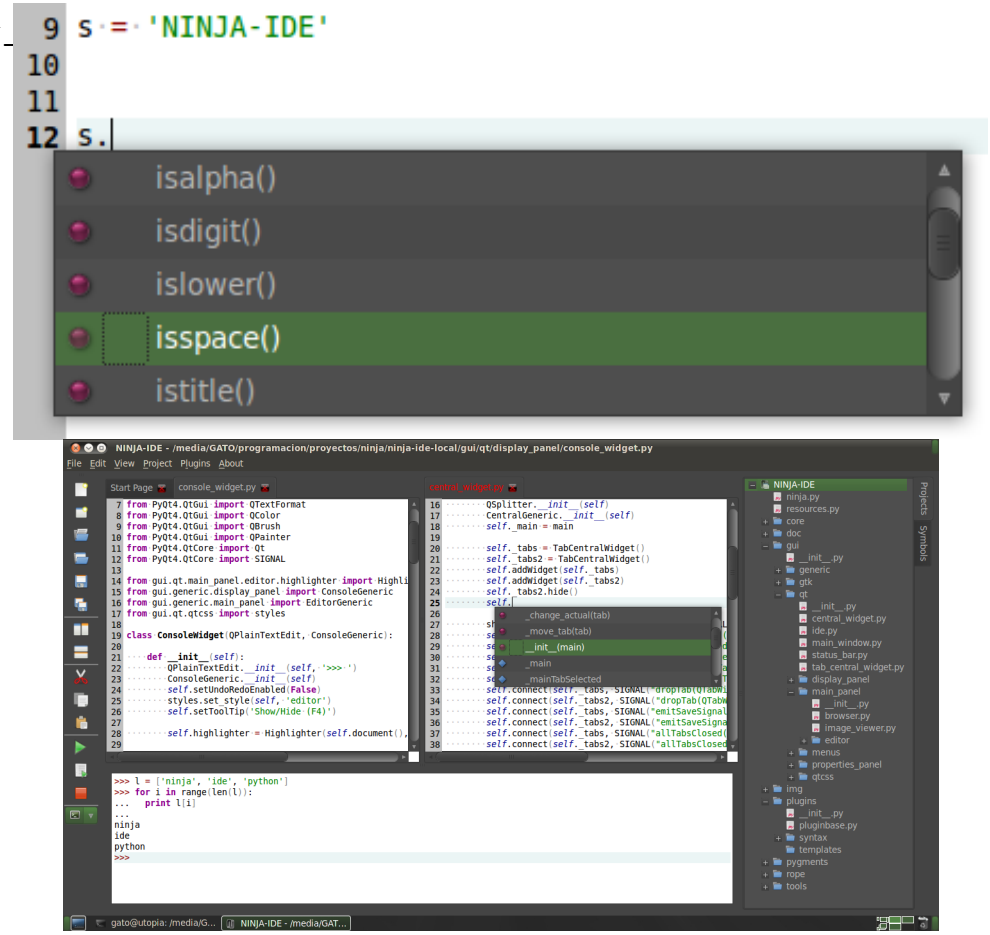
Cómo decidimos que características agregar?

Al comenzar el proyecto se pensó en una estructura que le permitiera al mismo crecer e incorporar funcionalidades a lo largo del tiempo, teniendo como guía dos factores principales: el Editor de Código y el Manejo de Proyectos. La aplicación comenzó a construirse cuidando estos dos pilares fundamentales y permitiendo que una buena base de los mismos facilitara luego la incorporación de nuevas características.

El proyecto fue pasando por distintas etapas, comenzando por un buen editor con resaltado de sintaxis, siguiendo con el manejo de archivos de proyecto, hasta agregar características de plugins, auto-completado, manejo de sesión, etc.

Muchas veces se ve a Python como un lenguaje que presenta mayores dificultades para brindar información sobre el código que se esta escribiendo al no poder hacer inferencia de los objetos en el momento de la programación a causa del tipado dinámico, etc. En algunos casos, es cierto que al contar con un tipado explicito se pueden realizar análisis más simples y detallados, pero también es cierto que actualmente existen muchas herramientas y librerías para Python que ayudan a eliminar este tabú de que no es posible contar con un IDE que brinde real asistencia sobre el código que se esta generando. Es por eso que NINJA-IDE busca permitir que aquellos programadores que utilizan Python para desarrollar sus programas, cuenten con las mismas facilidades y ayudas que se obtienen al desarrollar en Java o .NET con alguno de los IDEs más conocidos actualmente para esos lenguajes. Tomando los resultados y experiencias obtenidas de IDEs para otros lenguajes, se pretende lograr un IDE pensado para Python que genere la misma satisfacción al usarlo.

Para la sugerencia, decisión e incorporación de nuevas características en NINJA-IDE se suele utilizar la lista de correo para lograr una decisión colectiva por parte de los miembros que componen el proyecto, más que nada para saber cual será el objetivo de esta característica, en que etapa debería incorporarse y demás detalles. Muchas veces estas características son motivadas por alguna funcionalidad interesante vista en otro IDE, una idea de alguno de los miembros o sugerencias del grupo de usuarios. De este modo, cualquier persona, tanto usuario como desarrollador, puede plantear que cosas le gustaría ver implementadas en NINJA-IDE y en base a la arquitectura del proyecto se podrá definir si es necesario incorporarla como parte del IDE mismo o como un plugin, permitiendo a la vez conocer que ideas se están trabajando y quienes asumen el control de las mismas para mantener al grupo de trabajo sincronizado.



Que esperamos de NINJA-IDE?

NINJA-IDE nace para cubrir una necesidad que nos parecía importante, y además veíamos que los enfoques actuales de los IDEs no brindaban la cobertura necesaria. Nuestra intención al iniciar este proyecto fue crear un entorno centrado en el desarrollo de aplicaciones Python, pero siempre teniendo en cuenta la necesidad de contar con una comunidad de usuarios que nos permitiera mejorar la experiencia de uso de esta herramienta, y actualmente nos pone muy contentos poder estar contando con la comunidad de NINJA-IDE, ya que gracias a la experiencia y conocimiento colectivo de los usuarios es posible, que con sus sugerencias, el desarrollo del proyecto pueda avanzar más rápido y se tengan en cuenta muchos más detalles que de otra forma podrían ser pasados por alto.

Planes para el Futuro

Actualmente nos encontramos en la versión 1.1 de NINJA-IDE, la cual recibe la denominación de 'Kunai'. En esta primera versión están presentes las características mencionadas previamente, las que permitirán al desarrollador contar con un IDE robusto y práctico, obviamente como en todo proyecto irán surgiendo mejoras y nuevas features para implementar.

Algunas de las características que están pensadas para ser incorporadas en la versión 2.0 de NINJA-IDE, la cual acaba de comenzar su desarrollo, son:

- Implementación de una Arquitectura dinámica y extensible.
- Debuguer Gráfico.
- Poder ver la navegabilidad y relación de los módulos y clases de un proyecto de forma gráfica (basado en BlueJ).
- Soportar herramientas de versionado de código.
- Permitir la edición colaborativa de un documento.
- Diseñador de interfaces Qt integrado en el IDE.
- Localizador de Código.
- Soporte para Virtualenv.
- Agregar más features de Refactoring.
- Plugins para Versionado.
- Soporte para Frameworks como:
    - Django
    - Google App Engine
- Y esto apenas está comenzando!

Se puede consultar las Release Notes de las versiones actuales en: http://code.google.com/p/ninja-ide/wiki/ReleaseNotes Y un completo listado de las Features propuestas para la Versión 2.0 en: http://code.google.com/p/ninja-ide/wiki/Brainstorming_version2

Qué herramientas utiliza NINJA-IDE?

El IDE es desarrollado utilizando las librerías de PyQt para todo el manejo de la interfaz gráfica y algunas otras funcionalidades. Qt permitió contar con una interfaz solida y altamente configurable, lo que hizo posible poder extender de cada elemento necesario para modificar su comportamiento y adecuarlo a las necesidades del IDE.

En cuanto al resaltado de sintaxis, NINJA-IDE hace uso de su propio sistema de resaltado de sintaxis utilizando funcionalidades de Qt, y permitiendo que este sistema de resaltado sea fácilmente extensible en NINJA-IDE con la creación de un simple archivo JSON que describa al lenguaje que se desea incorporar. Este método brinda mejoras en la performance, pero para cubrir aquellos lenguajes que no sean reconocidos a través de este sistema se incorporó el uso de Pygments para el resaltado de sintaxis de una mayor variedad de lenguajes. Aunque para la versión 2.0 del IDE se tiene pensado cubrir la gama de lenguajes soportados por Pygments directamente con NINJA-IDE para mejorar la performance del resaltado de sintaxis y eliminar esta dependencia con una librería externa.

Para las funcionalidades de auto-completado, refactoring, y aquellas que se refieren a la inferencia del código, se utiliza Rope, la cual es una excelente librería, muy completa para este tipo de situaciones. Rope es una herramienta que permite llevar a un IDE para Python características de IDEs de lenguajes tipados.

Actualmente también es soportado el Chequeo de código utilizando la librería de Pep8, justamente para brindar información acerca del estado del código en relación a las normas de la Pep8.

Extensibilidad de NINJA-IDE

NINJA-IDE cuenta con un sistema de plugins bastante completo que permite la integración de dichos complementos como un elemento nativo del IDE. La escritura de Plugins es bastante sencilla y hasta se puede utilizar un Plugin de NINJA-IDE para la escritura de Plugins para NINJA-IDE (recursivo?). Este Plugin “para escribir Plugins” permite decidir con que partes del IDE el nuevo complemento se va a relacionar y crea de forma automática la estructura del proyecto necesario, junto al descriptor del Plugin para que NINJA-IDE lo pueda interpretar y la clase base de ese Plugin con los métodos que serán necesario reimplementar, a su vez, al terminar con la escritura del Plugin nos permite empaquetarlo para luego poder distribuirlo.

Actualmente existen 3 Plugins para NINJA-IDE disponibles:

- Pastebin: el cual permite enviar código a pastebin.com y devuelve el link resultante para poder compartir ese código.
- PluginProject: el encargado de crear proyectos Plugins para NINJA-IDE como mencionábamos.
- ClassCompleter: completa de forma automática algunas estructuras mientras se esta escribiendo código Python, como por ejemplo: crear el constructor de forma automática realizando la llamada a las Clases Padre que sean necesarias, etc.

Para consultar mayor información acerca de como desarrollar un Plugin para NINJA-IDE, se puede visitar la siguiente Wiki: http://code.google.com/p/ninja-ide/wiki/CrearPlugins

Usando librerías adicionales y virtualenv

Author: Tomas Zulberti

Los temas sobre los que voy a hablar son:

- La librería estándar de Python
- ¿Qué hacer cuando algo no esta en en la librería?
- Instalando librerías adicionales
- Usar virtualenv para solucionar los problemas que aparecen

Python viene con cerca de 350 librerías para usar. http://docs.python.org/library/index.html

Tienen de todo:

- Trabajar con mails
- Fetchear paginas
- Interactuar con el sistema operativo
- Trabajar con archivos comprimidos
- Parsear diferentes tipos de documentos (xml, json, etc..)

MUCHAS COSAS MAS....

Antes que nada, veamos algunos ejemplos de cosas que se pueden hacer con las pilas que trae Python.

Enviamos un mail usando una cuenta de GMail

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

>>> USERNAME = "tzulberti@gmail.com" # Aca tienen que poner su usuario
>>> PASSWORD = "no se los voy a decier" # Aca ponen su passsword
>>> mailServer = smtplib.SMTP('smtp.gmail.com',587)
>>> mailServer.ehlo()
>>> mailServer.starttls()
>>> mailServer.ehlo()
>>> mailServer.login(USUARIO, PASSWORD)
>>> msg = MIMEText(" Este es el contenido del mail... ")
>>> msg['From'] = USERNAME
>>> msg['To'] = USERNAME
>>> msg['Subject'] = "Este es el sujeto"
>>> mailServer.sendmail("tzulberti@gmail.com", "tzulberti@gmail.com", msg.as_string())
```

Leemos una pagina web para leer el contenido de la misma.

```
>>> from urllib2 import urlopen
>>> response = urlopen("http://python.org.ar/pyar/")
>>> contenido_html = response.read()
>>> print contenido_html
<!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>
```

Trabajamos con archivos comprimidos

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

Pero la librería de python no tiene todo:

- No sabe leer algunos algunos formatos de archivos (yaml, mp3, vídeo)
- Frameworks web (pylons, django, web2py)
- Usar API de Youtube, Deliciosus, etc....

Vimos que la librería estándar puede hacer varias cosas, pero hay cosas que son mas fáciles si se usan librerías hechos por otras personas:

- Los frameworks permiten ahorrar cierto trabajo repetitivo.
- Las interfaces a la API nos ahorran el hecho de tener que leer la documentación de las mismas
- Aunque Python sabe leer archivos de texto y binarios, es mas fácil que directamente convierta ciertos formatos de archivos a estructuras de python.

¿Dónde buscar librerías que no esten en python?

http://pypi.python.org/pypi

Hay cerca de 11.000 librerías para usar. Como se vio en la charla de Roberto Alsina (import antigravity), hay para hacer de todo:

- Colorear la consola
- Alternativas a la consola de python (bpython, ipython)
- Trabajar con formatos de archivos raros
- Bajar archivos de cuevana
- Bajar videos de youtube
- Parsear xmls usando una sintaxis parecida a jQuery
-
- MUCHO MAS COSAS

¿Cómo instalamos los paquetes de PyPi?

Primero necesitan instalar python-setuptools

Esto depende del sistema operativo:

- Ubuntu:

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

- Windows: tienen un .exe a bajarse desde acá:

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

Los paquetes en Python generalmente se distribuyen bajo un archivo **.egg**. Estos archivos son un archivo zip (con otra extensión) que:

- Tienen el código
- Tienen otros archivos (imagenes, sonido, docuementación, etc..)
- Tienen un archivo muy importante de metada: **setup.py**
- Para instalar los **.egg** necesitan instalar primero un modulo llamado setuptools.

Una vez que instalan el modulo **setuptools**, van a tener disponible el comando easy_install.

```
easy_install nombreLibreria
```

Por ejemplo:

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

Algo importante a tener en cuenta es que el comando easy_install también instala todas las dependencias. Por ejemplo, rst2pdf depende de Pygments y reportlab. Si se ejecuta:

```
easy_install rst2pdf
```

Entonces también se van a instalar reportlab, y Pygments. Para los que usan Ubuntu, easy_install funciona como **apt-get**.

Existe una alternativa a **easy_install** llamada **pip**. Recomiendo usar esa opción.

```
easy_install pip
```

Se puede hacer todo lo que se hacia con **easy_install**, y también hace mas cosas:

- Hacer búsquedas en pypi

```
pip search rst2pdf
```

- Desinstalar paquetes

```
pip uninstall rst2pdf
```

- Listar las librerías instaladas

```
pip freeze
```

- Instalar todas las paquetes que está en un archivo

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

Por ejemplo:

- pip install django
- pip install -U rst2pdf

Sin embargo existen algunos problemas:

- Es necesario ser admin/root. Es fácil cuando uno esta desarrollando, pero poco probable en producción.
- No se pueden instalar dos versiones de la misma paquete.

  Esto es importante cuando la paquete no es compatible para atrás. Algunos ejemplos son:

  - SQLAlchemy
  - django

  Por ejemplo, las aplicaciones que funcionan con django 0.9.7 no funcionan con django 1.2, y viceversa. Por como funciona el sistema de paquetes de Python, el mismo solo permite tener instalado una única versión de un mismo paquete.

  Por lo tanto, uno tiene que elegir que versión del paquete usar.

Usando virtualenv

Para solucionar estos problemas existe **virtualenv**. El mismo se encarga de crear un entorno aislado del entorno de Python del sistema. Para eso es necesario instalarlo:

```
easy_install virtualenv
```

Es una única vez que van a necesitar ser root/admin para instalar una librería. Lo traen todos los servidores que usan permiten hostear proyectos en python.

Sino se tiene permiso de root/administrador, también se lo puede usar sin instalarlo en el sistema. Para eso se bajan el comprimido desde:

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

Una vez instalado, lo primero que hay que hacer es crear un entorno virtual:

```
virtualenv nombreEntorno
```

Esto va a crear una carpeta **nombreEntorno** en donde estemos parados. Si no lo instalaron en el sistema, y se bajaron el archivo comprimido, entonces

```
python virtualenv.py nombreEntorno
```

El comando anterior creo un entorno de Python que esta separado de la configuración del sistema global. Por lo tanto, uno hay no tiene problemas de permiso porque esto lo puede crear en su carpeta.

Para decirle al sistema operativo que queremos usar el entorno creado, y no el del sistema, es necesario activar el entorno.

Linux:

```
source nombreEntorno/bin/activate
```

Windows:

```
nombreEntorno\Scripts\activate.bat
```

Tanto en windows como en linux, cuando activen en entorno, les va a aparecer el nombre del entorno activado entre paréntesis:

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

Cuando uno tiene en entorno activado, uno puede ver que el python que usa no es el global, sino que es esta dentro de la carpeta que creo **virtualenv**, y no el del sistema global

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

Lo mismo pasa con el **easy_install** y **pip**. También, todos los paquetes que se instalen se van a instalar dentro del entorno creado.

Para dejar de usar el entorno lo que tenemos que hacer es:

Linux:

```
deactive
```

Windows:

```
nombreEntorno\Scripts\deactivate.bat
```

¿Cómo funciona virtualenv?

Cuando uno crea un entorno, dentro del mismo se crean tres carpetas:

- bin: cuando uno tiene el entorno activado, y ejecuta un comando como *python*, *pip* o *easy_install*, se ejecuta alguno de los binarios que se encuentran en esa carpeta, en vez de ejecutar los del sistema.
- include: es simplemente un link al a los archivos de la instalación de Python.
- lib: esta es otra carpeta importante. Al igual que **include** tiene una carpeta llamada python pero a diferencia de **include**, esta no es una link a la carpeta de Python. La carpeta tiene dos cosas importantes:
    - Un link a algunos archivos de Python. En este caso a los .py
    - Una carpeta, **site-packages**, que es donde se *instalan* los paquetes cuando uno usa **pip** o **easy_install**

¿Preguntas y respuestas? (Parte I)

Anteriormente comente que uno de los problemas que teníamos era que no se podían instalar dos versiones de la misma librería. Lo que nunca hice fue comentar como se soluciona ese problema.

La solución es fácil. Se crea un entorno para usar con django 0.97, y otro para usar con 1.2. Se pueden crear todos los entornos que uno quiera, y son aislados entre si.

Para los servidores de producción, hay configuraciones se puede configurar el apache para que use cierto entorno virtual.

¿Preguntas y respuestas? (Parte II)

Creo un virtualenv, lo activo, y hago **pip freeze**, y me aparecen cosas que nunca instale en el entorno.

El comportamiento predeterminado de virutalenv cuando se crea un entorno, es también crearlo con los paquetes que uno tiene instalado en el sistema. Para que no haga eso, cuando se crea el entorno, se puede hacer:

```
virtualenv --no-site-packages nombreEntorno
```

Eso hace que cree un entorno totalmente vació. Por último, otra opción a tener en cuenta es **—python**, que hace que el entorno virtual use esa versión de python (para eso es necesario tener instalado esa version de Python). Por ejemplo, en mi maquina tengo instalado Python 2.7 y 3.1, siendo el 2.7 el predeterminado.

Luego, cuando se cree un entorno, lo va a crear usando Python 2.7. Para que en el entorno se use Python 3.1, se tiene que hacer:

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

¿Preguntas y respuestas? (Parte III)

Hay ciertos paquetes que tienen código escrito en **C**. ¿Qué pasa con las mismas?

En esos casos pip o easy_install van a intentar compilar el código escrito en **C**, cuando uno lo instala. En distribuciones como Ubuntu, recomiendo instalar:

- build-essential
- python-dev

Esos dos paquetes facilitan mucho la compilación.

Sin embargo, en Windows no es tan feliz la cosa. Varios de los paquetes que tienen código en **C**, se distribuyen también en **.exe**. Sin embargo, los instaladores no permiten seleccionar donde instalar las mismas. Acá `http://www.developerzen.com/2010/09/23/the-complete-guide-to-setting-up-python-development-environment-on-windows/` hay una MUY buena guia de como hacer para que compile los paquetes cuando los baje.

¿Preguntas y respuestas? (Parte IV)

¿Recomendas instalar los paquete del sistema o usando easy_install?

Hay que tener en cuenta que varias distribuciones traen varios paquetes de python que se pueden instalar en el sistema. Por ejemplo:

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

Pero las versiones que se pueden instalar por **apt-get** están desactualizadas. Acá vemos algunos ejemplos comparativos de versiones:

Nombre del paquete	Versión de Ubuntu 10.10	Versión de Pypi
rst2pdf	0.14.2	0.16
reportlab	2.4.3	2.5
django	1.2.3	1.2.3

Por lo tanto, depende de lo que uno quiera hacer. Por ejemplo, si uno quiere hacer un programa que funcione con Ubuntu, entonces puede crear un entorno virtual e instalar las mismas versiones que con **apt-get** o directamente instalar las global

Lo que **NO** recomiendo es sobre escribir la versión instalada. Por ejemplo, si uno hace:

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

Entonces no tiene que hacer:

```
easy_install -U reportlab
```

Sin usar un entorno virtual porque puede romper algo del sistema.

Desafío PET

Autor: Juanjo Conti y Alejandro Santos

En el número anterior de PET presentamos nuestro primer desafío. Consistía en proveer el programa más corto capás de factorizar un número (teniendo en cuenta algunas reglas). Hemos recibido muchas respuestas, las primeras con alrededor de 500 caracteres, y las últimas que fueron exprimiendo código byte a byte para lograr la solución más corta.

De los 58 participantes que tuvimos, estamos orgullosos de contarles que tuvimos 3 ganadores. Más aún, ellos, con su solución de 111 caracteres, llegaron al programa ganador por distintos caminos, recortando, masajeando y perfeccionando sus soluciones originales.

Los ganadores

Sin más preámbulos, los ganadores, junto a sus obras, son:

Chema Cortes con 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 con 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 con 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)
```

Mención especial

En esta oportunidad hemos considerado dar el premio a la entrada más tramposa a Darni, que en lugar de que renombremos su archivo pet1-darni.py pidió que mantengamos su seudónimo artístico *ZT1pbnB1dCgpCnM9JycKZD0xCndoaWxlIGU+MToKCXA9MDtkKz0xCgl3aGlsZSBlJWQ8MTplLz1kO*

Este es el contenido del archivo pet1-ZT1pbnB1dCgpCnM9JycKZD0xCndoaWxlIGU+MToKCXA9MDtkKz0xCgl3aGlsZSBlJWQ8MTplL

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

Información adicional sobre la competencia puede encontrarse en http://python.org.ar/pyar/Proyectos/RevistaPythonComunidad/PET1/Desafio

Nuevo desafío

El nuevo desafío fue escrito por Alejandro Santos, quien adora resolver problemas, y esta semana estuvo mandando varios Quizz a la lista de PyAr.

Gracias a Matias “Tuute” Bellone por leer una revisión inicial del problema.

El número QQQ

QQQ es la red social más popular de país del país de Ainohtyp. Al estilo microblog, QQQ es una red social donde los usuarios pueden publicar solamente la palabra QQQ. Los usuarios también pueden tener de contactos a otros usuarios, pero se tienen que aceptar mutuamente.

Los administradores de la red social armaron un listado con los usuarios ordenado por la cantidad de contactos de cada uno, y el Rulo encabeza la lista teniendo la mayor cantidad de contactos.

Tu trabajo es hacer un programa para averiguar cuál es la mínima distancia entre cada persona con el Rulo. Eso es, la mínima cantidad de saltos que hay entre el Rulo y el resto. Si hay dos o más formas de llegar al Rulo, elegir la más corta.

Por ejemplo, si el Rulo es amigo de Juan y Pedro, Pedro es amigo de Carla, y Carla es amiga del Rulo, Carla tiene distancia 1 con el Rulo, y Juan y Pedro tienen ambos distancia 1 con el Rulo.

Es posible llegar desde el Rulo a Carla por medio de Pedro, pero como Carla es amiga del Rulo la mínima distancia es 1. Pero si Jorge es amigo de Pedro, entonces para llegar desde el Rulo a Jorge la mínima distancia es 2 ya que se tiene que pasar por Pedro.

Datos de entrada

Todos los datos se leen por entrada estándar.

Por cuestiones de privacidad los datos son anónimos, y la única información de los usuarios es su número de usuario dentro de la red.

En la primer línea hay dos números N y M que dicen la cantidad de usuarios y la cantidad de relaciones que hay en la red social, respectivamente.

Los usuarios se numeran de forma consecutiva y sin descartar ningún usuario desde el 1 en adelante, siendo Rulo el usuario número 1. En las M líneas siguiente hay dos números que indican una relación entre dos usuarios.

Algunos usuarios tienen cero contactos o no hay forma de armar la cadena de contactos al Rulo.

Datos de salida

Imprimir por salida estándar N lineas con dos números separados por un único espacio en blanco. El primer número es el número de usuario, y el segundo número es la mínima distancia hacia el Rulo.

En caso de no poder calcular la distancia con el Rulo, mostrar el caracter “X”.

Ejemplo

Entrada:

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

Salida:

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

Los participantes deben enviar su solución como un archivo .py (con la forma pet2-USERNAME.py) y esta será ejecutado con python 2.7 en la computadora de Alejandro. El ganador del desafío será aquel que logre la solución que pase una batería de pruebas especialmente confeccionada en el menor tiempo posible.

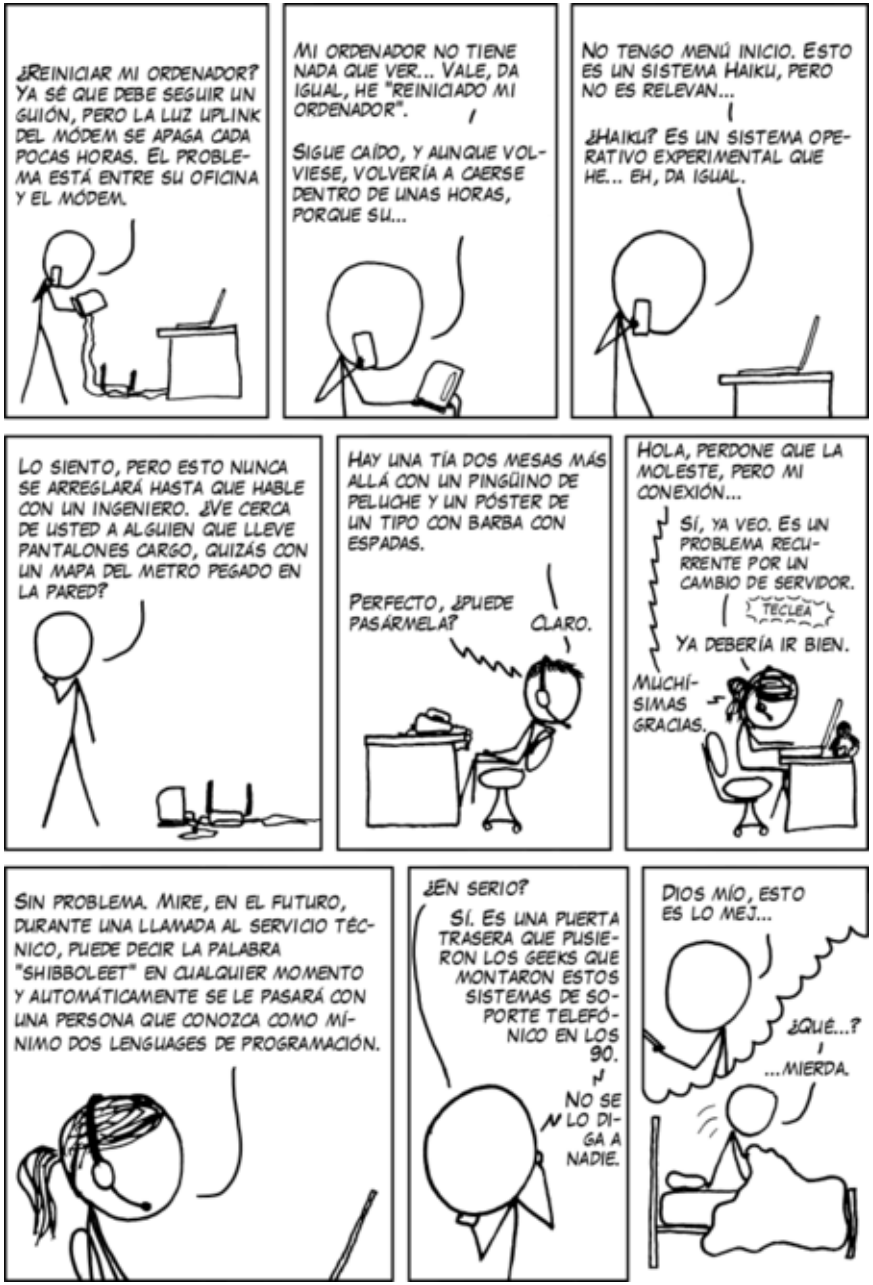
Enviá tu solución a revistapyar@netmanagers.com.ar poniendo `DESAFIO2` en el título del mail antes del 30/05/2011.

Suerte y happy brain squshing!

Aclaraciones y Feedback

Eventuales aclaraciones y feedback para los participantes será dado a través de la wiki: http://python.org.ar/pyar/Proyectos/RevistaPythonComunidad/PET2/Desafio

xkcd



Soporte Técnico

Este cómic proviene de xkcd, un comic web de romance, sarcasmo, matemática y lenguaje (http://xkcd.com)