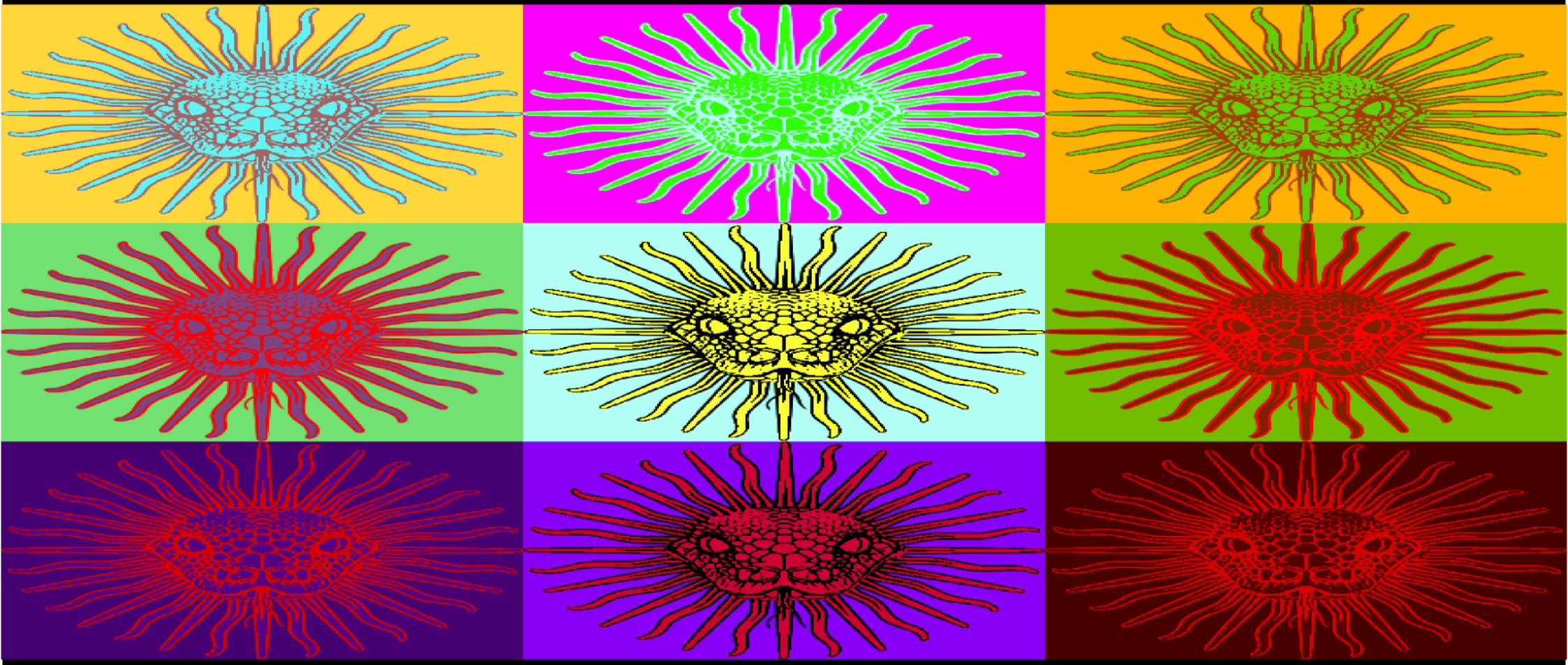
PET: Python Entre Todos

4 (sept 2011)



Revista de la comunidad Python Argentina

http://revista.python.org.ar/ ISSN: 1853-2071

Licencia

En Este Número

Licencia 2

PyTurismo 1

Asiru: linux para desarrollo con python 2

From gc import commonsense - garbage collection 5

Haystack - Buscando la aguja en el pajar 10

Jython: Mezclando Java y Python 15

Magic Mistery Forum 20

MATH2PY 25

RAD2PY: plataforma Python para el Desarrollo Rápido de Aplicaciones bajo un Proceso de Software Personal 30

Vida y obra de objetos persistidos en ZODB 38

xkcd 41

Staff

Editores: Juan Bautista Cabral - Tomas Zulberti

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

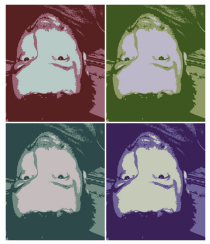
Los artículos son (c) de sus respectivos autores, reproducidos con autorización. El logo “solpiente” es creación de Pablo Ziliani.

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

PyTurismo



Autor: Juan B. Cabral

Bio: 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.

Web: http://jbcabral.wordpress.com

Twitter: @juanbcabral

Desde la edición anterior de la revista hasta esta que están teniendo en sus manos/monitores pasaron tres meses y 3 conferencias python en argentina: La primera fue el Django Day Córdoba 2011, la segunda fue el Pyday Gonzales Catán 2011 y la tercera fue el Pyday San Luis 2011.

Tuve la oportunidad de asistir a todas y sinceramente uno queda satisfecho con el empuje que pone la gente PyAr a estos eventos para lograr la difusión de la tecnología. Ojala esto no frene nunca.

Juan B Cabral Editor Pet Nro.4

Los disertantes del Pyday Gonzales Catán 2011

Quien les escribe y Matías Herranz (PyAr) en San Luis

Asiru: linux para desarrollo con python

Autor: Juan Pedro Fisanotti

Bio: Desarrollador python y django pero con gustos variados como ruby y lisp. Entusiasta de linux y el soft libre en general.

Email: fisadev@gmail.com

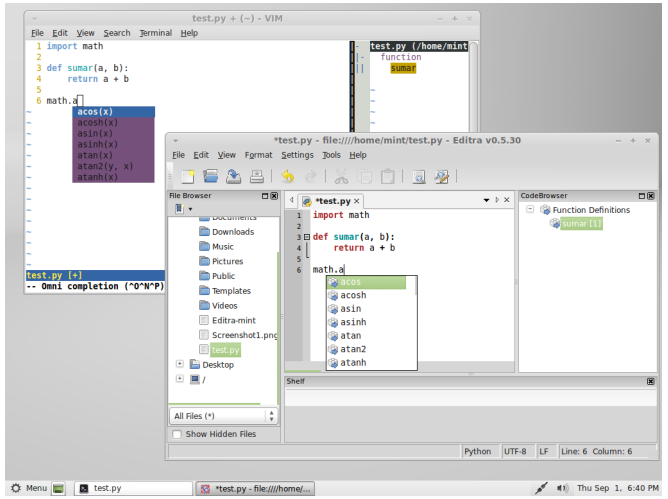
Twitter: @fisadev

¿Qué es Asiru?

Asiru es un Linux que trae ya instaladas y configuradas unas cuantas herramientas para desarrollar con Python, y está pensado para ser usado como LiveDVD (es decir, no para instalarse sino para ejecutarse desde un DVD o pendrive). ¿Qué tipo de herramientas incluye? IDEs y editores, sistemas de control de versiones, consolas interactivas, debuggers, bibliotecas y frameworks, etc. Detallo un poco más adelante.

Está basado en Linux Mint 11 (que a su vez está basado en Ubuntu) y se encuentra aún en etapa de desarrollo, aunque ya con una versión funcional. Nació para suplir algunos casos de uso bien específicos, pero con flexibilidad para crecer y adaptarse a otras necesidades.

Se trata de un proyecto que surgió casi por casualidad, pero que ahora parece tener ganas de crecer :). En esta nota les comento lo principal del proyecto, suficiente para que se interesen si les sirve, y no se aburran si no les sirve.



Surgimiento y objetivo (breve, para no aburrir)

Asiru surje de dos eventos diferentes que se cruzaron por casualidad:

Por un lado, hace un tiempo mi trabajo me llevó a entretenerme un poco (léase: renegar y divertirme) creando CDs de Ubuntu customizados. La necesidad era poder iniciar Ubuntu desde un CD, pero que el mismo ya posea determinadas aplicaciones instaladas y configuradas, más algunas personalizaciones al sistema operativo.

Encontré algunos tutoriales bastante detallados en la wiki de Ubuntu respecto a crear LiveCDs customizados, y a partir de ellos y de lo que algunos de mis compañeros ya habían hecho, logré el objetivo.

No fue sencillo, pero en el camino aprendí algunas cuantas cosas interesantes de Linux y de la magia que hay detrás de un LiveCD.

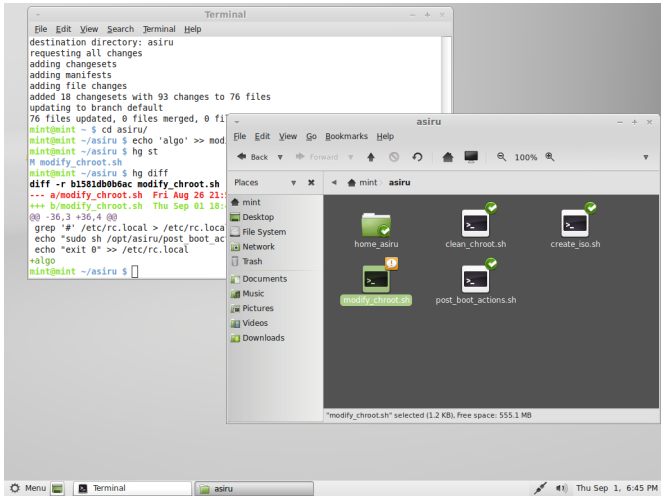
Por otra parte comencé a dar clases en una segunda materia de la universidad, ayudando en la parte de práctica (la materia es Inteligencia Artificial).

En esta materia la práctica se hace con Python como lenguaje de programación, y no siempre los alumnos conocen las herramientas que pueden utilizar para programar con él. Sumado a ello, la mayoría de los alumnos no han tenido previo contacto con Linux, lo que es una pena ya que no “elijen” qué sistema operativo utilizar, porque nadie les ha dado a conocer otra alternativa. Se pierden de conocer un sistema operativo que en mi opinión, se adapta mucho mejor a las necesidades de un desarrollador de software.

Viendo esta realidad y con la experiencia que había tenido en el trabajo, se me ocurrió que podría llegar a ser una buena idea proveer a los alumnos de un entorno de Linux ya preparado con las herramientas para desarrollar en Python. De esa manera no se pierde tanto tiempo en enseñarles a descargar y configurar estas herramientas y podemos pasar casi de lleno a la práctica, y al mismo tiempo tienen la posibilidad (porque no es obligación) de comenzar a conocer el sistema operativo que tantos preferimos para trabajar con Python, y para uso diario.

Ese fue el objetivo inicial de Asiru, pero a mitad de camino descubrí que podría resultar útil para otros casos de uso, como llevar en el bolsillo (pendrive) un entorno de desarrollo listo para usar en “emergencias” donde falte nuestra pc, o facilitar la difusión de Python a principiantes en el lenguaje distribuyendo con facilidad DVDs en conferencias o eventos.

Con estas cosas en mente decidí hacer de Asiru un proyecto libre y no restringirlo más a la necesidad de la que surgió, sino abrir las puertas a ideas y necesidades nuevas de quienes estén interesados.



Estado actual

Como dije antes, el proyecto se encuentra en desarrollo pero ya posee una versión usable. En http://asiru.googlecode.com se pueden encontrar los links para descargar la última versión de la imagen de DVD (que también puede ser utilizada para crear pendrives booteables), el código fuente que permite crear dicha imagen, y por el momento solo un poquito de documentación (estoy trabajando en ampliarla).

La versión funcional actual de Asiru inicia con las siguientes herremientas listas para usar:

Editores / IDEs:

- Vim y GVim: editor por linea de comandos, con mi configuración de vim (http://fisa-vim-config.googlecode.com), que incluye cosas como un debugger, autocompletado, navegador de clases, manejo de tareas pendientes, etc.
- Editra: editor con entorno gráfico con varios plugins instalados y configurados para autocompletado, navegación de clases, control de versiones, manejo de proyectos, etc.
- WinPDB: no es un editor, sino un debugger muy bueno para Python.
- Meld: un muy buen editor gráfico de diffs.

Sistemas de control de versiones:

- Mercurial: sistema distribuído de control de versiones, con la aplicación TortoiseHg para realizar control de versiones desde el entorno gráfico, y algunos plugins activados (hgk, graphlog, colors).
- Git: sistema distribuído de control de versiones.
- SVN: sistema centralizado de control de versiones.

Consolas interactivas de Python:

- iPython: consola con autocompletado, ayuda mejorada, y varias utilidades, y que se asemeja mucho en comportamiento a la mayoría de las consolas de Linux.
- bPython: consola con autocompletado y ayuda mejorada, con aspecto más “gráfico”.

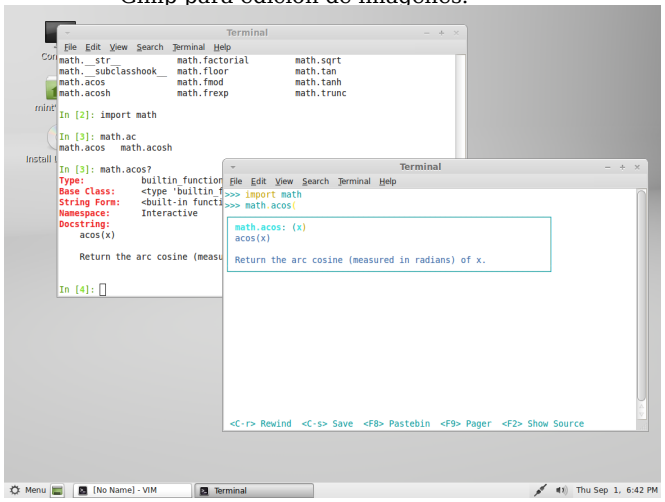
Bibliotecas y Frameworks:

- Django: framework de desarrollo web.
- Bottle: microframework de desarrollo web.

- VirtualEnv: para crear entornos virtuales de Python con sus propias bibliotecas y configuraciones.
- PyLint: biblioteca para análisis de código fuente de Python, que detecta errores comunes o signos de baja calidad en el código.

Otras cosas útiles:

- PIP: herramienta para la instalación simple de bibliotecas de Python.
- Gunicorn: servidor web simple para aplicaciones Python.
- Navegadores web Chromium y Firefox.
- Visor de VNC.
- LibreOffice para edición de documentos y presentaciones.
- Gimp para edición de imágenes.



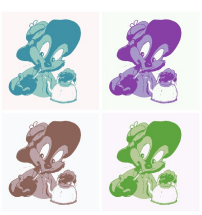
Planes a futuro

Seguramente a cada persona que lea la lista de las herramientas que hoy vienen en Asiru (y más si es un desarrollador Python con experiencia) se le van a ocurrir varios “¿cómo puede faltar XYZ?”. A mi ya se me ocurren algunos como Eclipse con PyDev, Bazaar, Twisted, Web2py, Ninja-IDE, etc.

Lo se, faltan muchas cosas importantes. Pero el tiempo y un poco de falta de conocimiento me limitaron la selección. Por eso estoy abriendo el juego para que quienes quieran puedan aportar lo necesario para que la lista se acerque más a lo que cada uno necesita en su entorno de desarrollo.

Si tienen ganas de pispear, o mejor, de aportar algo, péguense una vuelta por http://asiru.googlecode.com, que todavía hay lugar para unos cuantos. Se trata de un proyecto interesante y útil para más de uno en el mundo de Python, y una buena oportunidad de jugar con un poco de “magia negra” de linux :)

From gc import commonsense - garbage collection



Autor: Claudio Freire

Bio: Python, C, C++, Java, assembler, código máquina, lo que quieras.

Email: freireclaudio@yahoo.com.ar

Nadie piensa en cómo se administra la memoria en python. Es una de sus magias más queridas. Simplemente anda. Pero la verdad es que, cuando se pone presión sobre el colector de basura (garbage collector para los que saben) de Python, se vislumbran sus muchas limitaciones. Es imprescindible conocerlas para no encontrarse con problemas inesperados en nuestros programas.

Algo de contexto

Garbage collection, o recolección de basura, es cualquier método para detectar y liberar memoria no referenciada en un programa de manera automática. Hay muchos métodos de recolección de basura, aunque en general el término se asocia a los métodos de seguimiento de referencias utilizados por java, que siguen las cadenas de referencias desde las variables del programa para liberar cualquier objeto no alcanzable desde ellas.

Estos métodos tienen la ventaja de ser “a prueba de fallas”, en el sentido que son siempre efectivos en encontrar objetos que ya no son útiles para el programa. Pero carecen de inmediatez, pues deben realizar un análisis de todos los objetos en el sistema antes de poder detectarlos, y carecen de semánticas de finalización claras e inambiguas.

En Python, estos métodos no son aplicables. Debido a la arquitectura del código C de Python, no es posible saber todas las variables del programa pues, además de las variables que mantiene el intérprete, las bibliotecas de extensión pueden mantener referencias en la pila de C, que yace en el hardware y es difícil de explorar portable y eficientemente.

Desde su nacimiento, Python utilizó otro método, que es el conteo de referencias. A pesar de no llamarse “recolección de basura” en la literatura, es también un método de recolección de basura. Pero no es completo: el método falla si hay referencias cíclicas. Un ejemplo sencillo de referencia cícilica es un objeto que se referencia a sí mismo, como:

```
l = []
l.append(l)
del l
```

En el código de arriba, la lista nunca será liberada. El conteo de referencias funciona manteniendo un contador con la cantidad de punteros que apuntan a un objeto. Cuando la lista se crea, el contador es 1 (por la variable “l” que apunta al objeto). Al agregarse la lista a sí misma, el contador se vuelve 2 (porque la lista tiene internamente otro puntero que apunta a ella). Al limpiarse la variable (con “del l”), el contador vuelve a 1, pues la variable “l” ya no apunta a la lista, pero el puntero que la lista tiene apuntando a sí misma aún existe y cuenta como una referencia. El contador nunca volverá a cero, y por tanto la lista nunca será liberada.

En un entorno con conteo de referencias, estos ciclos se consideran mala práctica y, de hecho, casi siempre son evitables. El problema es que en programas complejos, a veces es difícil evitarlos, resulta en código antinatural, o incluso el programador puede no darse cuenta que está introduciendo ciclos, pues no siempre son tan directos como en el ejemplo. Un ejemplo más retorcido sería:

```
class SomeGarbage:
    def __init__(self):
        try:
            self.dosomething()
        except:
            self.error = sys.exc_info()
```

Muchos frameworks hacen este tipo de cosas, para poder reportar correctamente errores que surgen de improvisto sin romper la ejecución del programa. El tema es que exc_info incluye tracebacks, y los tracebacks incluyen referencias a todas las variables de la pila en toda la pila de llamadas. Esto incluye el self de SomeGarbage.__init__, y probablemente otras variables de código que llamó al SomeGarbage. En síntesis, el ciclo puede venir de código remoto y desconocido para los programadores. Es a veces difícil eliminarlos si no son tan evidentes.

Por eso, para Python se introdujo otro tipo de colector de basura, uno que opera sólo sobre los ciclos (que es el punto débil del conteo de referencias). Con este parche, introducido activado por defecto desde Python 2.0, un tiempo atrás, los programadores casi no tienen que preocuparse por nada. Casi.

Colector de ciclos

El algoritmo de Boehm-Demers es uno de los algoritmos de marcado y barrido (mark and sweep, como el que usa Java) más usados. El algoritmo primero marca todos los objetos como potencialmente no alcanzables, y recorre las referencias en la pila y las variables locales, desmarcando objetos a medida que encuentra referencias.

En el caso de python, este algoritmo se aplica únicamente a los ciclos. El objetivo es, pues, complementar al contador de referencias, detectando el tipo de basura que el conteo no puede detectar. Esto se llama pues colección de ciclos.

Como el contador de referencias puede manejar perfectamente todos los casos que no generan ciclos, esto elimina efectivamente cualquier objeto que no pueda contener referencias a otros objetos. Esto incluye multitud de tipos de objetos muy usados en python: números, cadenas, unos cuantos tipos de la biblioteca estándar, como slices, xrange, etc... todos tipos que deben ser pequeños y rápidos, siguen siéndolo si se ignoran en el colector de ciclos.

El colector de ciclos trabaja de forma inversa al colector de basura, pero utiliza el mismo algoritmo de Boehm-Demers, con pocas modificaciones. Tomando todos los objetos contenedores (es decir, listas, diccionarios, tuplas, objetos con atributos, etc), el algoritmo detecta los objetos que tienen referencias fuera de este conjunto. Esto es, referencias desde el intérprete o desde bibliotecas de extensión. Así que los objetos referenciados desde no contenedores, o alcanzables desde no contenedores, se consideran vivos. Todo objeto que sólo sea alcanzable desde contenedores y no desde afuera, debe ser, pues, basura.

Efectivamente, este método detecta todos los objetos python inútiles. Pero no siempre puede liberarlos.

Finalización de ciclos

Cuando los ciclos contienen finalizadores (`__del__`), surge un problema: no hay manera obvia de finalizarlos. Si dos objetos en un ciclo tienen finalizadores, cualquier orden de destrucción puede traer problemas. Si A se destruye primero, B podría intentar accederlo luego, o viceversa. Incluso si se llamara al finalizador sin destruir el objeto, los objetos pueden no estar preparados para el subsiguiente acceso puesto que se supone que el finalizador es lo último que se hace con un objeto.

En síntesis, es un caso imposible, y Python optó por considerarlos “Basura no colectable”.

La basura no colectable es un problema eterno para los programadores de python, difícil de diagnosticar y de solucionar una vez diagnosticado. Son objetos que python no puede liberar, y como quedaron fuera del alcance del programa, el programa tampoco puede hacerlo. Es, efectivamente, la única forma de memory-leak en python. En python puro, al menos.

Generaciones, guarderías y casas de retiro.

Como el análisis este es complejo y requiere recursos, surgió el conecpto de guarderías y generaciones. Las generaciones son grupos de objetos que han vivido por cierta cantidad de tiempo. En general, hay mínimo tres generaciones: la guardería, la generación donde nacen todos los objetos, la generación intermedia, y la generación permanente donde están los objetos de larga vida.

Las generaciones existen para acelerar la recolección de basura. La guardería es donde están los objetos de muy corta duración, se recolecta seguido, y suele ser pequeña. Cuando los objetos sobreviven una recolección en la guardería, se mueven a la generación intermedia, que se revisa menos seguido. A medida que van sobreviviendo, van moviéndose de generación en generación, hasta que llegan a la última generación, la generación permanente, que se limpia muy poco seguido.

Python tiene normalmente sólo tres generaciones, y la frecuencia de recolección está dada por una métrica poco común: la cantidad de creaciones menos destrucciones. La razón de usar esta métrica es para evitar disparar recolecciones cuando el conteo de referencia funciona bien. Cuando no funciona, se presume, habrán muchas creaciones pero pocas destrucciones. Un efecto indeseable es que, a veces, esta heurística no funciona, y un par de objetos en un ciclo pequeño puede que consuman mucha memoria, no disparen la recolección, y la memoria no se libere a tiempo.

Defragmentación

Algunos recolectores de basura, como el de java, no sólo liberan objetos inservibles. También mueven los objetos que sí sirven, para asegurarse que la memoria usada es bien compacta. La utilidad de esto se deriva del hecho que el sistema operativo opera en páginas y segmentos, la memoria libre en el medio de una página o segmento no está libre para otros programas, sólo para Python.

Como las bibliotecas de extensión están hechas en C (de hecho, python también), y usan punteros directos a los objetos, Python no puede defragmentar la memoria. Si moviera un objeto de lugar, no tiene forma de actualizar todos los punteros que hacen referencia a ese objeto.

Por lo tanto, la recolección temprana de los ciclos es más importante en Python que en otros lenguajes con recolectocción de basura, pues evita la fragmentación de la memoria. Con esto en mente, a veces, luego de operaciones complejas que sabemos que generan

mucha basura, puede ser conveniente llamar explícitamente a `gc.collect`. Lo que nos lleva a:

from gc import glory

El colector de ciclos de python viene con un módulo, el módulo `gc`, que nos permite tunear, depurar, e incluso forzar una recolección.

Una herramienta indispensable para la depuración es `gc.garbage`, una lista que contiene todos la basura no recolectable (o, directamente, toda la basura si se pone al módulo en modo depuración).

```
>>> import gc,sys
>>> class SomeGarbage:
...     def __init__(self):
...         try:
...             self.dosomething()
...         except:
...             self.error = sys.exc_info()
...     def __del__(self):
...         print "me borré"
...     def dosomething(self):
...         raise RuntimeError, "No te creo"
...
>>> g = SomeGarbage()
>>> gc.garbage
[]
>>> del g
>>> gc.garbage
[]
>>> gc.collect()
10
>>> gc.garbage
[<__main__.SomeGarbage instance at 0x7fdfa98f0b90>]
>>>
```

En este ejemplo vemos cómo los tracebacks generan referencias cíclicas, y al usar `__del__` para imprimir cuándo se destruye el objeto, estamos inhibiendo la recolección del ciclo. También es visible el hecho que la recolección no es inmediata, y para el ejemplo tuvimos que forzar la recolección (con `gc.collect`, muy útil por cierto). También podemos ver que `gc.collect` recolectó 10 objetos, lo cual ilustra lo común que es generar ciclos, y lo difícil que es vislumbrarlos.

De hecho, podemos ver exactamente dónde está el problema:

```
>>> g = SomeGarbage()
>>> g.error
(<type 'exceptions.RuntimeError'>, RuntimeError('No te creo',), <traceback object at 0x7fdfa98f0d40>)
>>> t = g.error[2]
>>> t.tb_frame
<frame object at 0x793d00>
>>> t.tb_frame.f_locals
{'self': <__main__.SomeGarbage instance at 0x7fdfa98f0c68>}
```

En modo depuración, `gc` nos permite ver todos los ciclos en nuestro programa. Honestamente, no lo encontré muy útil en proyectos grandes, llenos de ciclos colectables. Pero, a veces, es el único recurso que queda para depurar.

```
>>> import gc
>>> gc.set_debug(gc.DEBUG_SAVEALL)
>>> l = []
>>> l.append(l)
>>> del l
>>> gc.collect()
1
>>> gc.garbage
[[[...]]]
```

Normalmente, la lista no aparecería como garbage. Pero al poner al colector en modo depuración, todos los ciclos son almacenados en `gc.garbage`, no sólo los no colectables. Es útil cuando nuestro programa debe correr en tiempo real, pues la recolección de ciclos introduce pequeñas pausas que pueden no ser aceptables. Con este modo, podemos atrapar los ciclos generados por el programa para ver cómo no generarlos, y con eso evitar las pausas por recolección de ciclos.

Referencias débiles

Una de las maneras más sencillas (y más correctas) de evitar referencias circulares, es mediante la utilización de referencias débiles.

Las referencias débiles, son referencias que no incrementan el contador de referencias. Cuando un objeto es destruido, todas las referencias débiles se trasnforman en `None`. Es muy fácil utilizar referencias débiles, el módulo `weakref` nos provee una forma sencillísima de usarlas:

```
>>> import weakref
>>> a = A()
>>> a.me = weakref.ref(a)
>>> a.me
<weakref at 0x7f203a675b50; to 'instance' at 0x7f203a6a64d0>
>>> a.me()
<__main__.A instance at 0x7f203a6a64d0>
>>> 2 # hace falta, para olvidar a la referencia "_" del intérprete interactivo
2
>>> del a
borróseme
>>>
```

Este ejemplo ilustra cómo las referencias débiles rompen ciclos, pues pudimos borrar al objeto sin problemas, y lo fácil que es promover una referencia débil a fuerte (llamando a la referencia débil), que es necesario para utilizar el objeto referenciado.

Recetario - Qué evitar

Los siguientes ejemplos muestran formas comunes de generar leaks de memoria. O sea, hay que evitar hacer lo que se hace allí, o, al menos, pensarlo muy bien antes de hacerlo:

Guardar lambdas con self

Esto genera una referencia circular puesto que el objeto lambda contiene una referencia implícita al self. Es especialmente malo si `SomeClass` tiene `__del__`. También hay que tener cuidado con otras variables libres que no sean self, conviene pensar cuidadosamente antes de hacer esto.

Evitar:

```
class SomeClass:
    ...
    def somefunc(self):
        ...
        self.atributo = lambda : self.haceralgo()
```

No hay problema, sin embargo, si el lambda no se almacena en ningún atributo o lista del objeto. Si sólo se pasa por parámetros, está todo bien. Incluso si se almacena en otros objetos no relacionados (siempre que `SomeClass` no tenga, incluso indirectamente, referencias al objeto que contenga el lambda).

Guardar tracebacks

Es muy peligroso guardarse tracebacks. Si hace absoluta falta, conviene convertirlos a string, o tener algún método que los borre del objeto cuando ya no hagan falta

Evitar:

```
class SomeClass:
    ...
    def somefunc(self):
        try:
            ...
        except:
            self.error = sys.exc_info()
```

Generadores

No es mala idea usarlos, pero hay que tener en cuenta que todas las variables locales del generador quedarán vivas mientras se use el generador. Cuando los generadores tienen variables libres, es exactamente lo mismo que un lambda, y pueden tener referencias a self y otras variables libres. Lo bueno, sólo referencian a variables que hayan sido usadas en el generador, así que es posible usarlos bien

Evitar:

```
def __init__(self, raw_data):
    self.raw_data = raw_data
    self.data = itertools.cycle(x for x in self.raw_data if x.enabled)
```

En este ejemplo, sería mejor referenciar directamente a raw_data - o incluso usar `@property` para definir el atributo `data`, en vez de almacenar el iterador directamente.

Recetario - Qué usar

Los siguientes ejemplos muestran formas comunes de evitar leaks de memoria.

Guardar lambdas con referencias directas

En vez de acceder a los atributos a través de self, siempre que se pueda, conviene referenciar al valor que tienen directamente. Por supuesto, si es necesario que el lambda referencie al valor actual al momento de ser llamado (y no al momento de ser creado), puede que esta técnica no funcione.

```
class SomeClass:
    ...
    def somefunc(self):
        ...
        algunvalor = self.algunvalor
        self.atributo = lambda : algunvalor
```

Guardar tracebacks formateados

Es muy peligroso guardarse tracebacks. Así que si hace falta, conviene guardarlos como strings.

```
class SomeClass:
    ...
    def somefunc(self):
        try:
            ...
        except:
            self.error = traceback.format_exc()
```

Generadores

Como con los lambdas, conviene apuntar a valores directamente. O no usarlos.

```
def __init__(self, raw_data):
    self.raw_data = raw_data
    self.data = itertools.cycle(x for x in raw_data if x.enabled)
```

O

```
def __init__(self, raw_data):
    self.raw_data = raw_data
    self.data = itertools.ifilter(operator.attrgetter('enabled'), self.raw_data)
```

O

```
def __init__(self, raw_data):
    self.raw_data = raw_data
```

```
@property
def data(self):
    raw_data = self.raw_data
    return itertools.cycle(x for x in raw_data if x.enabled)
```

Si hace falta usar generadores de la manera que generan ciclos, asegurarse que los objetos involucrados no tengan `__del__`.

Haystack - Buscando la aguja en el pajar

Autor: Matias Bordese

Bio: Computólogo y desarrollador Python/Django, fan del software y las tecnologías libres.

Web: http://matias.bordese.com.ar

Email: mbordese@gmail.com

IRC: matiasb (freenode)

Twitter: @mbordese

Realizar búsquedas sobre el contenido de un sitio es un problema habitual a resolver si uno hace desarrollo web. Sin embargo, para los que nos gusta trabajar con Python y Django es un problema para el cual este framework no provee una solución en sí mismo.

Afortunadamente existe una app que se presenta como una muy buena alternativa para encontrar lo que se busca, aunque sea una aguja en un pajar.

Conceptos previos

Antes de empezar a hablar del tema que nos convoca, algunas definiciones que es importante conocer:

- Motor de búsqueda (Search engine) Es el sistema que realmente va a resolver el problema de indexar y buscar la información; es el backend de la app de búsqueda, del cual nos abstrae. Existen diversas opciones: Solr, Xapian, Whoosh.
- Índice Es el almacenamiento utilizado por el motor de búsqueda. En general, la información guardada es no relacional y no sigue un esquema fijo (es decir, distinto a una base de datos relacional).
- Documento Es un registro en el índice. Se puede pensar como un diccionario. Suele contener un bloque de texto que sirve como contenido primario de búsqueda y campos con metadata adicional.

Haystack

Haystack[1], una app desarrollada por Daniel Lindsley, provee una solución modular para resolver el problema de la búsqueda en Django. En otras palabras y de manera simplista, es una capa de abstracción que permite integrar Django con un motor de búsqueda (como por ejemplo Solr, Whoosh, etc).

Una pregunta que uno podría hacerse es por qué trabajar en un sistema de búsqueda personalizado y no usar directamente Google. Y la verdad es que podemos encontrar ventajas no despreciables:

- Control sobre qué y qué no se indexa
- Mejor calidad de la información en el índice
- Manejo particular de ciertos datos
- Posibilidad de búsquedas específicas según el contexto

Reconocidos los beneficios, la próxima cuestión que se plantea es por qué deberíamos usar Haystack y no interactuar directamente con el motor de búsqueda, por ejemplo. Habiendo usado Django previamente, existen motivos varios por los cuales deberíamos considerar esta app como potencial solución a nuestro problema:

- API familiar para desarrolladores Django, parece Django
- Pluggable backends (soporta Solr, Whoosh y Xapian)
- El código es independiente del backend

- Se integra a las apps que queremos sean buscables sin necesidad de modificarlas
- Buena documentación
- Nivel de tests aceptable, no se aceptan nuevos commits sin tests

Quiero usar Haystack!

A continuación, la idea es desarrollar un ejemplo simple que muestre cómo usar Haystack en nuestros sitios Django. Para ello tomamos como base el siguiente modelo:

```
from django.contrib.auth.models import User
from django.db import models

class Entry(models.Model):
    title = models.CharField(max_length=100)
    created = models.DateTimeField(auto_now_add=True)
    author = models.ForeignKey(User)
    tease = models.TextField(blank=True)
    content = models.TextField()

    def __unicode__(self):
        return self.title
```

Una de las primeras cuestiones que nos va a interesar es la de poder indexar nuestros datos, hacerlos “buscables”.

En Haystack, lo que determina qué información se indexa son los SearchIndex [2]. De alguna forma se asemejan a un modelo de Django, o a un Form, en el sentido de que son field-based, y manipulan y almacenan datos.

En general, vamos a crear un SearchIndex por cada modelo que queremos indexar. Hay que tener en cuenta que, para realizar búsquedas complejas entre múltiples modelos, es conveniente definir los nombres de los campos apropiadamente, preservando dichos nombres para los diferentes modelos cuando corresponda.

Definamos pues nuestro SearchIndex. Para ello, creamos una clase que herede de SearchIndex, definimos los campos en los que queremos guardar datos y registramos esta clase asociándola a nuestro modelo (notar en este punto la similitud con el registro de ModelAdmins):

```
from haystack import indexes, site
from myapp.models import Entry

class EntrySearchIndex(indexes.SearchIndex):
    text = indexes.CharField(document=True, use_template=True)
    author = indexes.CharField(model_attr='user__username')
    created = indexes.DateTimeField()

    def index_queryset(self):
        return Entry.objects.published()

    def prepare_created(self, obj):
        return obj.pub_date or datetime.datetime.now()

site.register(Entry, EntrySearchIndex)
```

Este código se suele agregar en un archivo search_indexes.py dentro de la app, aunque no es requerido. Siguiendo la convención en el nombre, permitimos que haystack.autodiscover() lo encuentre y registre automáticamente[3].

Notar también que todo SearchIndex debe tener un único campo con el parámetro document=True, indicando que se trata del campo primario de búsqueda. Éste debe mantener el mismo nombre a través de los distintos modelos, por convención se lo llama text.

A dicho campo además podemos asociarle un template (use_template=True), lo que nos permite definir la información a indexar de manera más cómoda. Creamos entonces, en el directorio de templates del proyecto, el archivo search/indexes/myapp/entry_text.txt:

```
{{ obj.title }}

{{ obj.author.get_full_name }}

{{ obj.tease }}

{{ obj.content }}
```

Adicionalmente, agregamos a nuestro SearchIndex un par de campos más, útiles al momento de proveer filtros, o para contar con metadata adicional al presentar los resultados al usuario. Vale destacar que al realizar búsquedas a través de Haystack

estaremos trabajando sobre el índice del motor de búsqueda, no sobre la base de datos, e idealmente nos gustaría evitar los hits a la misma.

Haystack dispone de una buena variedad de SearchFields [4] para manejar la mayoría de los distintos tipos de datos. Usualmente bastará con pasar el parámetro model_attr para indicar el campo de nuestro modelo del cual provendrán los datos a indexar. Como argumento se puede pasar tanto un atributo como un callable de nuestro modelo.

A veces puede ser necesario un mayor control de la información a indexar. En ese caso, se puede definir una preparación de los datos, previa al indexado. En el ejemplo de arriba definimos el prepare_created, que preprocesa el campo created. Esto debería resultar familiar a la forma en que Django maneja la validación de campos en un Form (y sus métodos clean). De la misma forma, aquí podemos definir métodos prepare_FIELD(self, object); o un prepare(self,object), más general, permitiendo acceder a los distintos campos en self.prepared_data.

Finalmente, en nuestro SearchIndex hacemos un override del método index_queryset, que devuelve qué objetos indexar cuando se hace una actualización del índice.

Existen varios métodos más que podemos personalizar para adaptar el indexado de los datos a nuestras necesidades. Para más detalles pueden consultar en la documentación de Haystack, que se referencia al final del artículo.

El que busca, encuentra

Ahora que ya indexamos la información, sería bueno saber cómo encontrar la aguja en el pajar.

Para buscar sobre nuestro índice contamos con SearchQuerySet [5]. Esta clase está diseñada para efectuar búsquedas de manera eficiente e iterar sobre los resultados de una forma fácil y consistente. Para aquellos que usan frecuentemente los QuerySet del ORM de Django, la API de SearchQuerySet les resultará muy accesible.

SearchQuerySet provee una API limpia, que nos abstrae del motor de búsqueda que usemos como backend. Y al igual que con Django QuerySet, podemos encadenar los métodos de búsqueda uno después de otro para restringir los resultados.

Además, al implementar una interfaz list-like, podemos realizar operaciones para pedir la cantidad de resultados, acceder a los resultados mediante índices posicionales o incluso aplicar slices.

Dentro de SearchQuerySet distinguimos métodos que devuelven un nuevo SearchQuerySet (encadenables entre sí), y otros que no.

Entre los primeros podemos citar, por ejemplo: all, none, filter (soporta field lookups![6]), exclude, order_by, models. Entre los que no: count, best_match, latest. Cada uno de estos hacen lo que uno esperaría, y mapean casi directamente a sus equivalentes en Django QuerySet, a excepción de un par nuevos como models que permite filtrar por modelos, o best_match que devuelve el primero de los resultados encontrados.

Por otro lado, una de las situaciones más comunes consiste sin duda en buscar sobre nuestro campo primario de búsqueda. Para facilitar esto (al usuario y al backend), hay un campo especial llamado content que podemos usar en cualquiera de los métodos que esperan un nombre de campo (filter, exclude, etc) para indicar que queremos buscar sobre el campo que denotamos con document=True en nuestros SearchIndexes.

Llevando a la práctica lo anterior:

```
>>> import datetime
>>> from haystack.query import SearchQuerySet

# Buscamos en todos los fields marcados con `document=True`.
>>> results = SearchQuerySet().filter(content='hello')
[<SearchResult: myapp.entry(pk=u'3')>]

>>> sqs = SearchQuerySet().models(Entry)
>>> sqs = sqs.filter(created__lte=datetime.datetime.now())
>>> sqs = sqs.exclude(author='daniel')

# La query realmente se hace cuando listamos los resultados (lazy).
>>> sqs
[<SearchResult: myapp.entry (pk=u'5')>, <SearchResult: myapp.entry(pk=u'3')>,
<SearchResult: myapp.entry (pk=u'2')>]

# Podemos iterar sobre los resultados. No hiteamos la DB.
>>> [result.author for result in sqs]
['johndoe', 'sally1982', 'bob_the_third']

# Un hit a la DB por cada resultado.
>>> [result.object.user.first_name for result in sqs]
['John', 'Sally', 'Bob']

# Más eficiente, una sola query a la DB.
>>> [result.object.user.first_name for result in sqs.load_all()]
['John', 'Sally', 'Bob']
```

Como podemos observar los resultados devueltos son instancias de SearchResult [7], a través de la cual podemos acceder a la metadata indexada para cada resultado, y de ser necesario a la instancia de modelo correspondiente (atributo object).

Hay más

Haystack también provee views y forms simples[8], que cubren los casos más comunes y completan la implementación de todo lo necesario para integrar búsquedas a nuestro sitio.

Las views incluidas permiten: búsqueda básica por términos, búsqueda con filtros por modelos, búsqueda en cuyos resultados se destacan los términos de la consulta ("highligted search") y búsqueda facetada (filtros anotados con el número de resultados por cada valor posible).

No existe prácticamente acoplamiento entre views y forms, y podemos intercambiarlos libremente o definir los propios a nuestra medida.

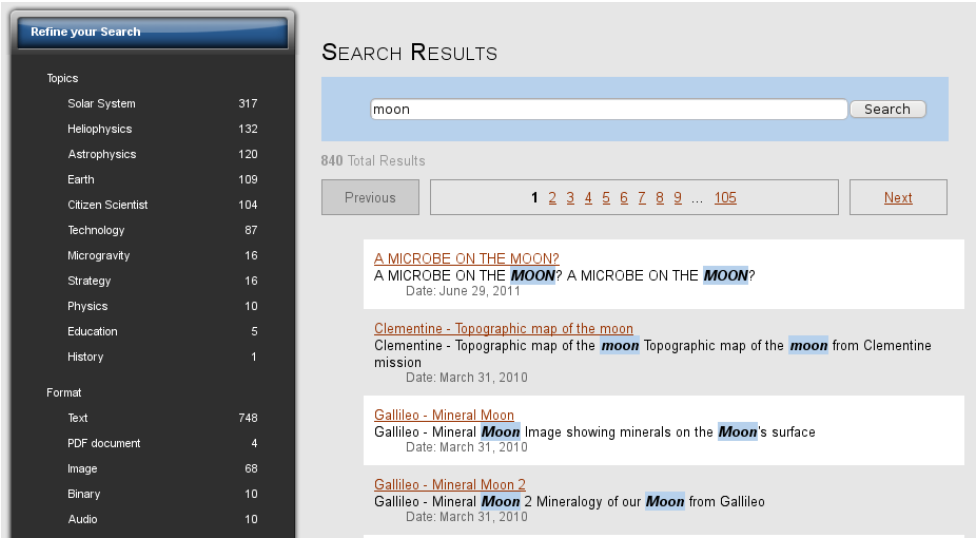
Además existen varias características más avanzadas que podemos integrar a nuestro sistema de búsqueda mediante Haystack:

- Highlighting[9]
- Faceting[10]
- Boost[11]
- More like this[12]

Finalmente, pero no menos importante: Haystack es Python y Django! Es decir que, si out-of-the-box no se ajusta a lo que necesitamos, podemos partir de SearchIndex y SearchQuerySet para obtener lo que queremos.

Para cerrar, algunos sitios que usan Haystack y que muestran algunas de las posibilidades que nos brinda esta app[13]:

- NASA: http://science.nasa.gov/search/?q=moon



- Todas las recetas: http://www.todaslasrecetas.es/receta/s/?q=langostinos

Los invito a probar Haystack y sacar sus propias conclusiones.

Referencias

[0] Charla DjangoDay: http://www.carpe-diem.com.ar/2011/07/05/django-day-cordoba/

[1] Haystack: http://haystacksearch.org

[1] Source: https://github.com/toastdriven/django-haystack

[2] SearchIndex API: http://docs.haystacksearch.org/dev/searchindex_api.html

[3] SearchSite settings: http://docs.haystacksearch.org/dev/tutorial.html#create-a-searchsite

[4] SearchFields: http://docs.haystacksearch.org/dev/searchfield_api.html

[5] SearchQuerySet API: http://docs.haystacksearch.org/dev/searchqueryset_api.html

[6] SearchQuerySet Field Lookups: http://docs.haystacksearch.org/dev/searchqueryset_api.html#id1

[7] SearchResult: http://docs.haystacksearch.org/dev/searchresult_api.html

[8] Views/Forms: http://docs.haystacksearch.org/dev/views_and_forms.html

[9] Highlighting: http://docs.haystacksearch.org/dev/highlighting.html

[10] Faceting: http://docs.haystacksearch.org/dev/faceting.html

[12] Boost: http://docs.haystacksearch.org/dev/boost.html

[12] More like this: http://docs.haystacksearch.org/dev/templatetags.html#more-like-this

[13] Sitios usando Haystack: http://docs.haystacksearch.org/dev/who_uses.html

Jython: Mezclando Java y Python

Autor: Milton Labanda

Bio: Ingeniero en Informática (Universidad Técnica Particular de Loja), Master de Software Libre (Universitat Oberta de Catalunya). Actualmente Técnico y Docente de la Carrera de ingeniería en sistemas en la Universidad Nacional de Loja. Coordinador de la Comunidad de Software Libre ESOL-UNL.

Web: http://1000tonlab.wordpress.com

Email: 1000ton.lab@gmail.com (http://revista.python.org.ar/1/html/mailto:1000ton.lab@gmail.com)

Twitter: @miltonlab

Identi.ca: miltonlab

Facebook: Milton Lab

Algunos desarrolladores, por no decir muchos, han coincidido en que “programar en Java no es lo mismo que programar en Python”, o viceversa, “programar en Python no es lo mismo que programar en Java”. Hoy en día gracias a proyectos como Jython o JPype se puede usar lo mejor de cada lenguaje en una misma aplicación.

Introducción

Los avances dentro del desarrollo de herramientas relacionadas con los diversos lenguajes de programación libres y/o de código abierto en estos tiempos, a mi criterio, están dando lugar a la programación colaborativa entre mas de un lenguaje a través de las diferentes liberías, implementaciones, APIs, bindings o como se las quiera llamar.

¿Que es Jython?

Uno los casos comentados en la introducción es Jython, una implementación de Java sobre Python. Empezamos entonces dando una definición formal, clara y concisa de lo que es y lo que permite Jython, de acuerdo al libro La Guia Definitiva de Jython de la Editorial Apress:

> ` “Jython es una implementación del lenguaje Python para la plataforma Java ... Jython trae el poder del lenguaje Python hacia la Maquina Virtual de Java. Provee a los desarrolladores Java la habilidad de escribir de escribir código productivo y dinámico usando una sintaxis elegante. Asi mismo permite a los desarrolladores Python ganar ventaja de la utilidad de las librerias y APIs que la JVM (Maquina Virtual de Java) tiene para ofrecer”. `

El Caso de Uso: Usando Swing desde Python

Uno de los campos donde python ofrece diversidad de posibilidades y alternativas es la creación de GUIs para aplicaciones de escritorio, en donde a través de los “bindigns” respectivos podemos usar desde Tk**(Tkinter) pasando por **wxWindows y llegando hasta Gtk o Qt u otras alternativas. Ahora bien si nos trasladamos al mundo Java nos encontramos en cambio con únicamente dos posibilidades AWT o SWING, aunque existe SWT parte del proyecto eclipse y no viene incluido en el API estandar de la plataforma.

Entonces a más de uno pudo haberle surgido la interrogante de: Si se puede usar Swing en o desde python?. La respuesta es si, es por ello que en este articulo nos ocuparemos de dar una visión considerable de la aplicabilidad que tienen el poder usar Jython para permitir la creación de interfaces gráficas que incluyan widgets o componentes de las librerías Swing perteneciente a la plataforma Java con la sintaxis de Python.

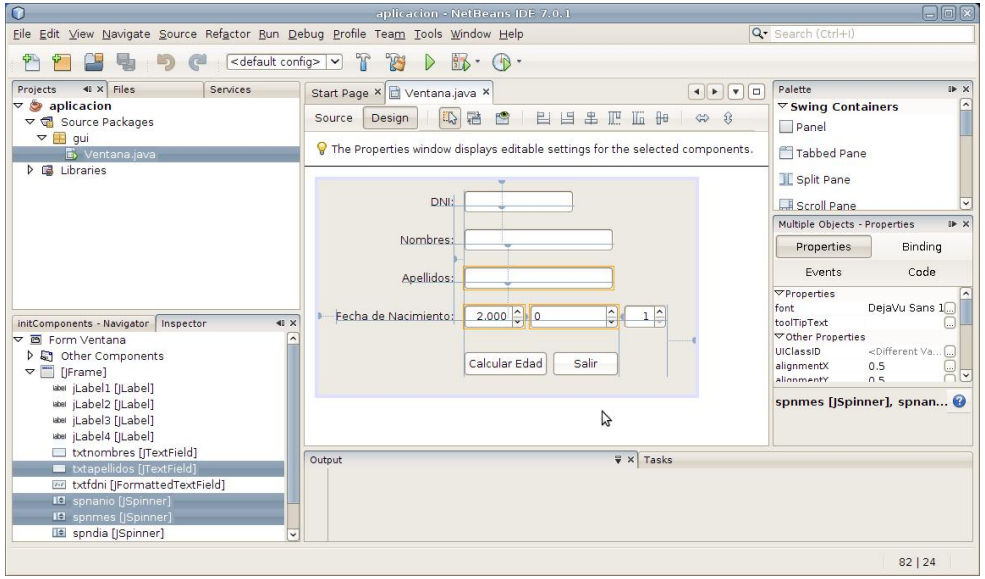
A pesar de que podemos construir íntegramente toda una aplicación usando código Jython, existe la interesante posibilidad y ventaja de crear la GUI totalmente en Java aprovechando de esta manera las herramientas e IDEs que existen para generación automática de Interfaces gráficas de usuario, tal es el caso de NetBeans o Eclipse con el plugin Visual Editor por ejemplo.

Construyendo la Vista del Ejemplo

Para ejemplificar el uso de Swing en una aplicación Jython hemos escogido algo muy simple como el cálculo de edad de una persona dada su fecha de nacimiento a más de calcular el tiempo exacto que falta para su próximo cumpleaños.

Aunque la temática de la interfaz gráfica al igual que la aplicación en general es de muchísima sencillez, hemos decidido usar el IDE Netbeans con el fin de ejemplificar las ventajas de poder generar con la ayuda de asistentes la GUI en un lenguaje y el resto de componentes de la aplicación en otro lenguaje de programación.

Veamos ahora creación de la Interfaz Gráfica de Usuario con el asistente de Netbeans:



Vale la pena señalar que aprovechando las novísimas características de la versión 7 de Java hemos fijado a Nimbus como tema de presentación de Swing (`javax.swing.plaf.nimbus.NimbusLookAndFeel`) incluido en esta ultima versión, con el siguiente código dentro de la clase creada con el designer:

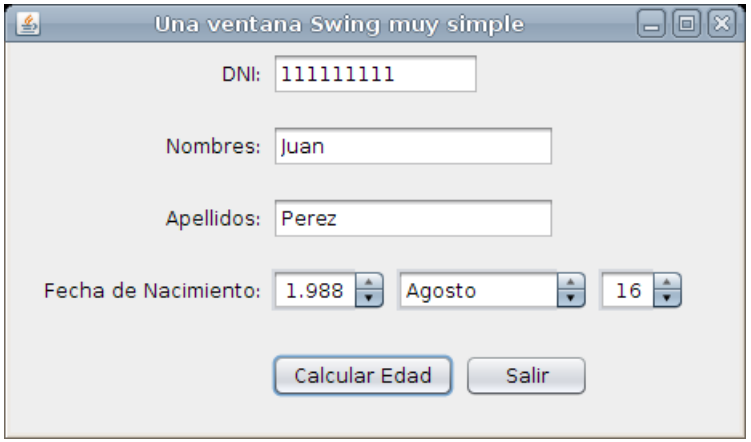
```
private void fijarLookNimbusJava7(){
        /* Set the Nimbus look and feel */
        //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">
        /* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.
         * For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
         */
        try {
            for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName())) {
                    javax.swing.UIManager.setLookAndFeel(info.getClassName());
                    break;
                }
            }
        } catch (ClassNotFoundException ex) {
            java.util.logging.Logger.getLogger(Ventana.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
        } catch (InstantiationException ex) {
            java.util.logging.Logger.getLogger(Ventana.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
        } catch (IllegalAccessException ex) {
            java.util.logging.Logger.getLogger(Ventana.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {
            java.util.logging.Logger.getLogger(Ventana.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
        }
}
```

Por otro lado es necesario definir como propiedades (con los métodos set y get) a los componentes de la GUI (mostramos algunos, pero el ide ayuda a crearlos a todos de una manera rapida y sencilla):

```
/**
 * @return the btncalcular
 */
public javax.swing.JButton getBtncalcular() {
    return btncalcular;
}

/**
 * @param btncalcular the btncalcular to set
 */
public void setBtncalcular(javax.swing.JButton btncalcular) {
    this.btncalcular = btncalcular;
}

/**
 * @return the btnsalir
 */
public javax.swing.JButton getBtnsalir() {
    return btnsalir;
}

/**
 * @param btnsalir the btnsalir to set
 */
public void setBtnsalir(javax.swing.JButton btnsalir) {
    this.btnsalir = btnsalir;
}

/**
 * @return the spnanio
 */
public javax.swing.JSpinner getSpnanio() {
    return spnanio;
}

/**
 * @param spnanio the spnanio to set
 */
```

```
public void setSpnanio(javax.swing.JSpinner spnanio) {
    this.spnanio = spnanio;
}
```

Una vista previa de la vista de la aplicación generada en la clase java gui.Ventana:



Programando la lógica de la aplicación en Python

Para generalizar la solución del problema relacionado con el cálculo de la edad, hemos reducido la lógica central a una resta de fechas de acuerdo a nuestro calendario. Antes de mostrar el código en Python mostraremo la lógica del algoritmo utilizado.

Aunque pueden haber muchas otras soluciones usando por ejemplo la clase `datetime.timedelta` hemos escogido usar un algoritmo que sea lo más generalizado para este tipo de situaciones. Así, si queremos saber el tiempo que hay entre las fechas 2011-08-12 y 1975-09-13, el procedimiento que aplicamos es:

```
 2011 - 08 - 12     31 - 1 = 30 (days)   Respuesta : 35 years, 2 months 30 days
 1975 - 09 - 13     12 - 2 = 2 (months)
   35 -(-2)-(-1)
```

Cuando el día es negativo aumentamos uno al mes de la fecha menor y lo restamos del numero total de días del mes anterior de la fecha mayor, en este caso 31 (por el mes 07 es decir Julio). Asi mismo si el mes resulta negativo le sumamos uno al año de la fecha mayor y lo restamos de 12 para obtener el mes real resultante.

Ahora el código en Python:

```
# Archivo: logic.py
def restarfechas(fecha1, fecha2):
    """
    Resta dos objetos datetime.date, el menor del mayor
    Devuelve una tupla con el numero de anios, meses y dias
    """
    f1,f2 = max(fecha1,fecha2),min(fecha1,fecha2)
    dias = f1.day - f2.day
    # se devuelve 1 a los meses en caso de ser negativos los dias
    f2 = f2.replace(month = f2.month + 1 if dias < 0 else f2.month)
    # numero de dias que tiene el mes anterior de la primera fecha
    if f1.month == 1:
        dias_mes_anterior = calendar.monthrange(f1.year-1,12)[1]
    else:
        dias_mes_anterior = calendar.monthrange(f1.year,f1.month-1)[1]
    dias = dias_mes_anterior + dias if dias < 0 else dias
    meses = f1.month - f2.month
    # se devuelve 1 a los anios en caso de ser negativos los meses
    f2 = f2.replace(year = f2.year + 1 if meses < 0 else f2.year)
    meses = 12 + meses if meses < 0 else meses
    anios = f1.year - f2.year
    return (anios, meses, dias)
```

De esta manera entonces reduciremos el cálculo de la edad y del tiempo del próximo cumpleaños prácticamente a una simple resta de fechas:

```
def edad(nacimiento):
    """ Devuleve la edad en una tupla de anios, meses y dias """
    hoy = datetime.date.today()
    return restarfechas(hoy,nacimiento)

def cumpleanios(nacimiento):
    """ Devuelve el tiempo restante para el siguiente cumpleanios """
    hoy = datetime.date.today()
    cumpleanios = datetime.date(hoy.year,nacimiento.month,nacimiento.day)
    if cumpleanios < hoy:
    cumpleanios = cumpleanios.replace(year = hoy.year + 1)
    faltan = restarfechas(cumpleanios,hoy)
    return (faltan[1],faltan[2])
```

Enlazando las partes de la aplicación con Jython

Finalmente enlazamos todo el escenario con un script en el cual básicamente creamos los manejadores Swing o AWT para que procesen los eventos y las interacciones del usuario interconectando con la lógica ya programada en Python puro.

Mostramos la clase principal por decirlo así, en donde creamos la GUI y registramos los manejadores de eventos, evidenciando además el uso de las librerías de la JVM, lo cual puede ser interpretado por Jython:

```
# Archivo: aplicacion.py
from java.awt.event import ActionListener
from javax.swing.event import ChangeListener
from javax.swing import JOptionPane
from java.lang import Integer

import datetime, calendar

import gui, logic

class Aplicacion:

    form = None

    @staticmethod
    def run():
        Aplicacion.form = gui.Ventana()
        Aplicacion.form.visible = True
        Aplicacion.form.btnsalir.addActionListener(ManejadorSalir())
        Aplicacion.form.spnmes.addChangeListener(ManejadorMes())
        Aplicacion.form.btncalcular.addActionListener(ManejadorCalcular())
```

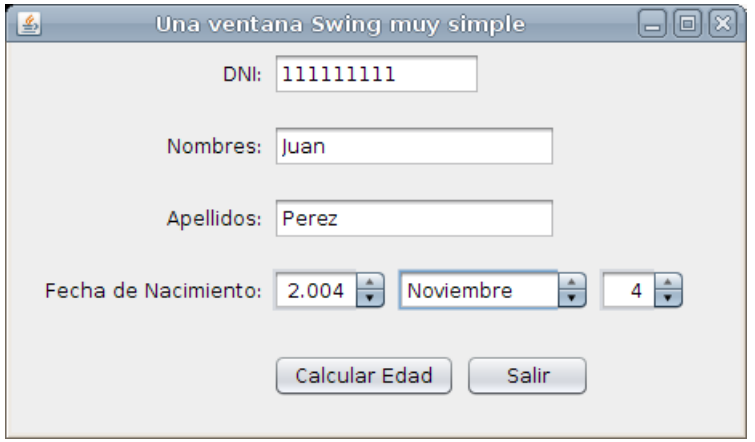
Para muestra el manejador de eventos que ejecuta el calculo de la edad:

```
class ManejadorCalcular(ActionListener):
    def actionPerformed(self,ev):
        mesint = Aplicacion.form.spnmes.model.list.indexOf(Aplicacion.form.spnmes.value) + 1
        fecha_n = datetime.date(Aplicacion.form.spnanio.value,mesint,Aplicacion.form.spndia.value)
        edad = logic.edad(fecha_n)
        nombre = '%s %s' % (Aplicacion.form.txtnombres.text,Aplicacion.form.txtapellidos.text)
        msg1 = '%s tiene %d anios %d meses %d dias' % (nombre,edad[0], edad[1], edad[2])
        JOptionPane.showMessageDialog(None, msg1)
        cumple = logic.cumpleanios(fecha_n)
        meses,dias = cumple[0],cumple[1]
        if meses == 0 and dias == 0:
            msg2 = 'Hoy es su cumpleanios.  Felicitaciones :)'
        else:
            msg2 = 'Faltan %d meses %d dias para su cumpleanios' % (cumple[0],cumple[1])
        JOptionPane.showMessageDialog(None, msg2)
```

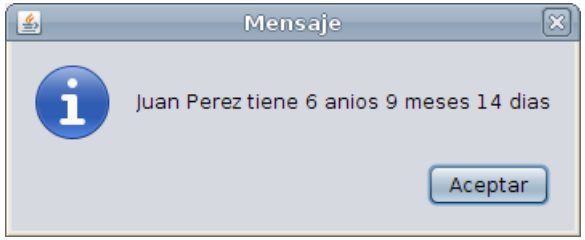
A continuación unas muestras de la ejecución de la aplicación (si la fecha actual es 18 de agosto del 2011) que se debe ejecutarse con:

```
miltonlab@debianlab:~/edadjpy$ jython aplicacion.py
```

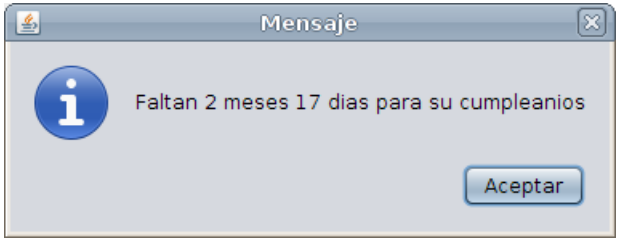
Ventana inicial



Presentación de la edad



Tiempo restante para el cumpleaños



Para aquellos que deseen revisar el código completo de la aplicación pueden encontrarlo en http://docencia-programacion.googlecode.com/files/edadjpy.tar.gz

Las versiones del software con el cual se ha desarrollado son: Java 6 (la versión 7 si se desea usar el Look And Feel “Nimbus’), Python 2.6.6 y Jython 2.5.1. El Sistema Operativo utilizado para la ejecución fue Debian 6.0

Magic Mistery Forum

Autor: Roberto Alsina

Bio: Gurú de PyQt, rst, y miembro de Canonical.

Webpage: http://lateral.netmanagers.com.ar/

Twitter: @ralsina

Hace un tiempo se juntaron muchos programadores Python en un mismo lugar, a tratar de divertirse programando. Yo estaba ahí, aunque no programé mucho, tuve una idea. De esa idea hay un video, que no pienso linkear porque soy gordo, estaba cansado y parezco Bellini el mentalista.

La idea era acerca de como se podría hacer una aplicación de foros sencilla. Muy sencilla. Demasiado sencilla. [7]

Partamos de definir exactamente qué es un foro.

1. Sitio web
2. Los usuarios pueden entrar con nombres distintivos.
3. Se pueden crear “hilos” de discusión.
4. En dichos hilos, los usuarios pueden responder a los comentarios de otros.
5. Los comentarios pueden verse instantáneamente en el sitio.
6. Algunos usuarios son moderadores, y pueden borrar spam, o comentarios agresivos, o contrarios a reglas de convivencia del foro.
7. Los usuarios tienen avatares, imágenes que los identifican.
8. Es posible suscribirse a un hilo o tema, y recibir notificaciones cuando alguien comenta en el mismo.
9. Podés postear imágenes, videos etc.

Tal vez, si nos ponemos demandantes podríamos pedir integración con redes sociales, o la posibilidad de votar mensajes o temas interesantes.

Bueno, vamos a hacer eso. Todo eso. Pero lo vamos a hacer de la mejor manera posible: haciendo trampa.

Para ello, vamos a utilizar los servicios de Disqus (http://www.disqus.com) un proveedor de sistemas de comentarios para sitios web (por ejemplo blogs) [8].

Resulta que Disqus ofrece un bonito API (http://disqus.com/api/docs/) y hay un módulo python para el mismo.

¿Cómo funciona? Poniendo un poquito de javascript en tu HTML. Ese javascript tiene un identificador de sitio y uno de discusión. Con eso, Disqus reconoce “ok, esta es la discusión ABC1 del sitio tal”, busca en su base de datos, genera un gran pedazo de HTML, lo injerta en tu página web, y listo, comentarios en tu página.

Ahora bien, nadie dice que la página que contiene los comentarios tenga que tener algo interesante aparte de los comentarios, ¿no?

Por otro lado, es posible obtener una lista de “todas las discusiones activas del sitio”.

Entonces podemos usar esa lista de discusiones para generar la “lista de temas” del foro.

Y cada tema podría ser un link a una página básicamente vacía, con el suficiente javascript como para que salgan los comentarios...

Empecemos con lo primero, una lista de temas generada a partir de una cuenta de Disqus.

Cualquier framework, miniframework, o microframework sirve. Yo voy a hacerlo con bottle (http://bottlepy.org) pero solamente porque ya sé como funciona. Podés usar Flask, Django, web2py, pyramid, o hacer un script cgi... es muy básico.

```
import time
# Necesito un framework web
import bottle
# Necesito la API de Disqus
import disqusapi as disqus
```

```
# Configuracion
shortname = 'magicmisteryforum'
url = "http://foro.netmanagers.com.ar"
# Esto lo obtenes vos en la página de disqus. Cada persona que quiera
# hostear esto necesita una distinta.
api = disqus.DisqusAPI(open("key").read().strip())

# Que vemos cuando el usuario entra en "/"?
@bottle.route('/', method='GET')
def index():
    # Parametro opcional "msg", para errores
    msg = bottle.request.GET.get('msg', '')

    # Tomamos la lista de threads de Disqus
    threads = api.threads.list(forum=shortname, limit=25)

    # Le pasamos esas dos cosas a un template
    return bottle.template('main.tpl', threads=threads,
        shortname=shortname, msg=msg, url=url)
```

¿Y qué es ese template? HTML medio feo, sabrán disculpar:

```
<h1>Magical Mistery Forum</h1>
<!-- Mostramos el error -->
%if msg:
    <div class="error">
    {{msg}}
    </div>
%end
</hr>
<div>
<!-- Un widget de disqus que muestra cosas interesantes -->
<script type="text/javascript" src="http://bit.ly/nsOAAA">
</script><a href="http://disqus.com/">Powered by Disqus</a>
</div>

<!-- formulario para crear nuevo hilo -->
<div>
<form method="POST" action="/new">
<span class="loud">New Thread Title:</b>
<input type=text name=title>
<input type="submit" value="Create">
<div class="error">When you create a thread, it may take a minute
to appear on the list.</br> All threads and posts will be deleted
every now and then, this is a test site only.
</div>
</div>
<!-- lista de todos los hilos existentes -->
% for t in threads:
    %if not t['identifiers'] or t['identifiers'][0] != t['title']: continue
    <div class="large">
    <a href="/thread/{{t['id']}}">{{t['title']}}</a>
    [ <a href="/thread/{{t['id']}}#disqus_thread"
    data-disqus-identifier="{{t['id']}}">#</a>]
    <div class="small">
      <small>{{t['likes']}} likes -- {{t['dislikes']}}
      dislikes -- {{t['createdAt']}}</small>
    </div>
    </div>
% end
<hr>
<!-- Comentarios -->
<script type="text/javascript">
    var disqus_shortname = '{{shortname}}';

    /* * * DON'T EDIT BELOW THIS LINE * * */
    (function () {
        var s = document.createElement('script'); s.async = true;
        s.type = 'text/javascript';
        s.src = 'http://' + disqus_shortname + '.disqus.com/count.js';
        (document.getElementsByTagName('HEAD')[0] ||
        document.getElementsByTagName('BODY')[0]).appendChild(s);
    }());
</script>
```

¿Y dónde te lleva lo de “crear nuevo hilo”?

```
# Se accede en la URL /new
@bottle.route('/new', method='POST')
def new():
    # El usuario manda el titulo del hilo
    title = bottle.request.forms.get('title', None)
    if not title:
        bottle.redirect('/?msg=Missing%20Thread%20Name')
        return
    # Creamos el hilo en disqus
    thread = api.threads.create(forum=shortname, title = title, identifier = title)
    print "THREAD", thread
    thread_id = thread.__dict__['response']['id']

    # Esto estaría buenísimo pero no anda porque hay una demora
    # en la creación del hilo
    #api.posts.create(thread=thread_id, message="Post about %s here!"%title)
    #bottle.redirect('/thread/%s'%thread_id)

    # Y te mandamos a la lista de hilos de nuevo, donde este aparecerá luego
    bottle.redirect('/')
```

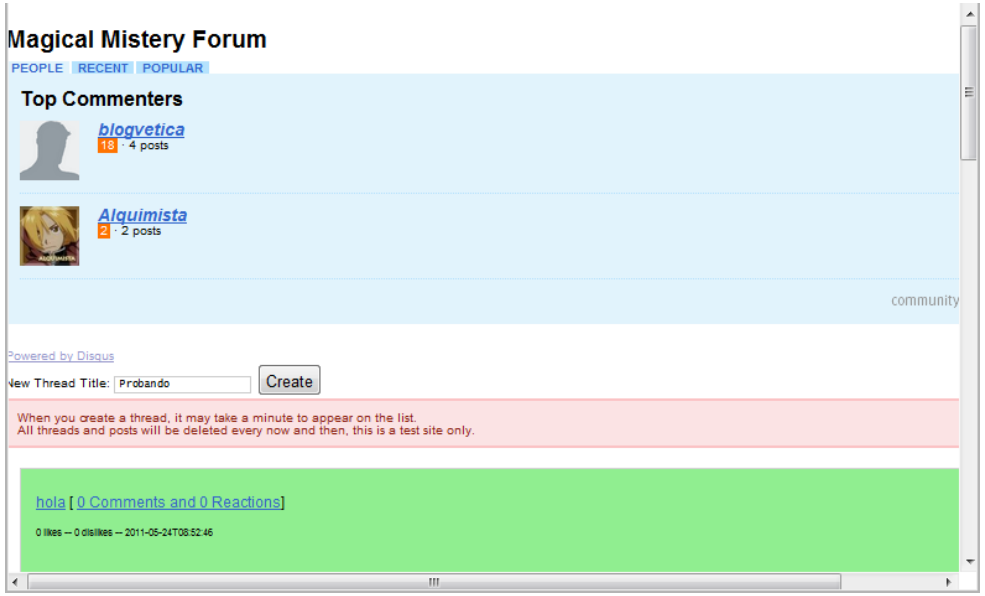
¿Y cómo se puede ver un hilo?

```
# El hilo esta en /thread/id
@bottle.route('/thread/:id')
def thread(id):
    t = api.threads.details(thread=id)
    return bottle.template('thread.tpl',
        shortname=shortname,
        id=id,
        thread=t.__dict__['response'])


# Codigo standard que no es muy interesante
@bottle.route('/static/:path#.+#')
def server_static(path):
    return bottle.static_file(path, root='./static')

app = bottle.app()
app.catchall = False
bottle.run(host='0.0.0.0', port=80, app=app)
```

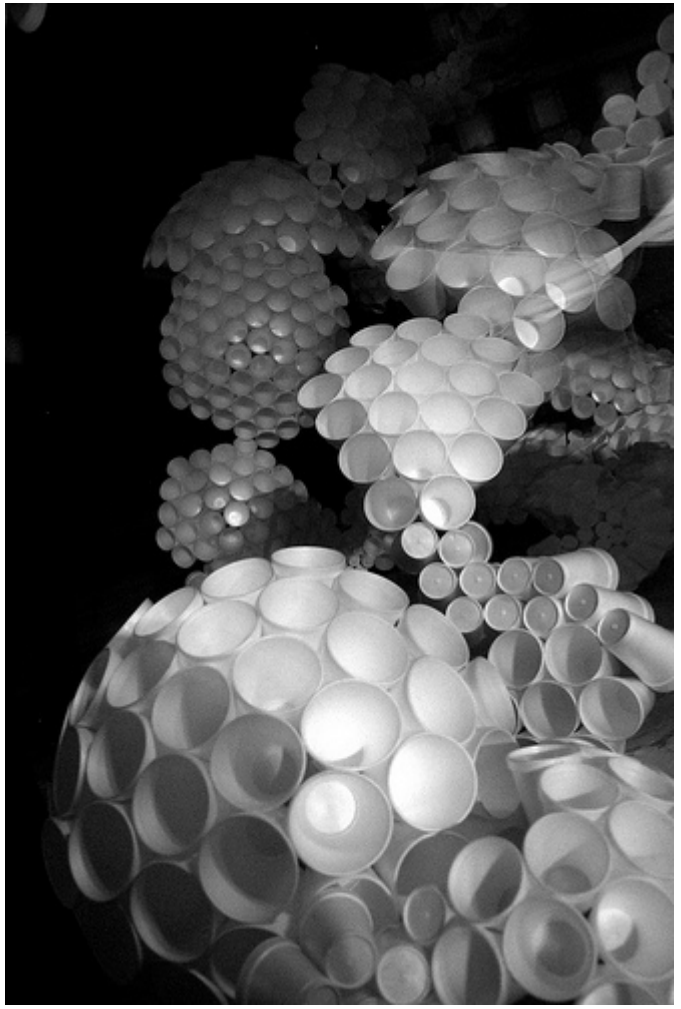
Faltaría ver el template que se usa para ver un hilo:

```
<!-- una manera de volver a la lista de threads -->
<div>
<a href="javascript:history.back(1)">Back to Thread List</a>
</div>
<!-- titulos -->
<h1>Magical Mistery Forum</h1>
<h2>{{thread['title']}}</h2>
<hr>
<!-- comentarios -->
<div id="disqus_thread"></div>
<script type="text/javascript">
    var disqus_shortname = '{{shortname}}';
    var disqus_identifier = '{{id}}';
    var disqus_url = '{{url}}/thread/{{id}}';
    (function() {
        var dsq = document.createElement('script');
        dsq.type = 'text/javascript';
        dsq.async = true;
        dsq.src = 'http://' + disqus_shortname + '.disqus.com/embed.js';
        (document.getElementsByTagName('head')[0] ||
            document.getElementsByTagName('body')[0]).appendChild(dsq);
    })();
</script>
<noscript>Please enable JavaScript to view
the <a href="http://disqus.com/?ref_noscript"> comments powered by
Disqus.</a></noscript><a href="http://disqus.com" class="dsq-brlink">
blog comments powered by <span class="logo-disqus">Disqus</span></a>
```

Y... no hay más aplicación. Eso es todo. Funciona? Sí!



Hasta ahora... un hack. Pero pensémoslo un poco más filosóficamente. Si bien no digo que esta chanchada es un foro maravilloso que reemplaza a phpBB o engendros similares, muestra como con un poco de maña uno puede tomar componentes provistos y usarlos para generar cosas distintas. Python, usado de esta manera, regresa un poco a sus orígenes como “glue language”, el lenguaje usado para pegar cosas heterogéneas y convertirlas en algo distinto.



073110 make believe project wicker park-17
(http://www.flickr.com/photos/kymberlyanne/4855669534/)

Y resulta que hacer “una aplicación web” usando un framework no es súper difícil [9]. Y hacer un deployment de una tampoco lo es. Y te da un campo para jugar. Y a veces, simplemente uno hace estas cosas porque... es divertido.

El código completo de Magic Mistery Forum está en http://magicforum.googlecode.com

7 Lamentablemente nunca llegué al extremo de simplicidad que pensé en ese momento. Tal vez algún día…

8 Son, por lo que oí, la aplicación Django mas grande del mundo.

9 En serio, el lado Python de la aplicación “se entiende” si sabés el lenguaje. ¿No?

MATH2PY

Autor: Alan Etkin

Bio: Técnico de PC y redes LAN, programador amateur de aplicaciones web en Python. Estudié Diseño en la UBA.

Web: http://spametki.blogspot.com

Email: spametki@gmail.com

Twitter: @spametki

Producción de una app/plugin web2py para herramientas matemáticas pythónicas

Descripción del proyecto

La idea básica de math2py es la de una interfaz web completa para el uso de librerías para matemáticas creadas en Python, instalable en web2py. math2py presenta un servicio de herramientas web para aplicaciones matemáticas y puede utilizarse como base de desarrollo en Python de subaplicaciones para uso técnico-científico de fácil instalación y personalización, con acceso a los paquetes de Python estándar y contribuciones.

La aplicación fue creada inicialmente como proyecto instalable (aplicación independiente) y luego modificada de modo que permita la instalación sobre el entorno web2py como aplicación o que se pueda incluir como accesorio de otra app central, en modo plugin [1]. La ventaja de la instalación como plugin es que un desarrollador o cualquier usuario con dominio de Python y web2py puede organizar las funcionalidades de la herramienta, agregando, quitando o modificando las subaplicaciones incluidas (componente según la convención para aplicaciones web2py [2])

Para facilitar y simplificar las condiciones de la instalación de la app, los paquetes, librerías y módulos para matemáticas incorporados en el dominio de la interfaz son de tipo Python puro. El motivo de esta elección es la usual restricción de acceso del intérprete Python a código C en ciertos servidores, como Google App Engine. La instalación de módulos de matemáticas en Python para la aplicación se ha limitado en cierto modo por esta restricción, y quedan afuera paquetes como sage, que es un recurso importante para la materia. Sin embargo, math2py es una aplicación para Python y es posible su adaptación para incorporar toda herramienta de matemáticas, como implementación alternativa y extendida de la versión estándar.

La aplicación incluye subaplicaciones de matemática básicas para resolución y presentación de problemas de álgebra lineal. Además incorpora interfaces para el cálculo y expresión gráfica de funciones.

¿Otra aplicación web matemática?

Las razones para utilizar math2py:

Es instalable en cualquier servidor compatible con web2py, incluyendo el modo de ejecución en desarrollo con la aplicación rocket server (ejecución en equipo local por puerto número 8000 o similar). De este modo, el uso del programa no se restringe a la instalación en red, y permite correrlo en una máquina local, por medio de un navegador web. El navegador web es entonces la pieza de soft principal utilizar math2py, desde la perspectiva del acceso básico a las herramientas. La forma aplicación web ó plugin requiere un lapso de tiempo de instalación despreciable entre descarga y carga mediante la interfaz admin de web2py.

Importa únicamente módulos de Python puro (por defecto), luego es posible utilizar cualquier librería Python (las restricciones de importación de módulos son limitaciones externas, en especial del servidor sobre el cual funciona web2py).

Solución open source, colaborativa, libre, sobre una plataforma de desarrollo web en crecimiento.

Permite el intercambio teórico y técnico por medio de notas y rutinas almacenables en la base de datos. Personalizable y ampliable. Una de las ventajas del desarrollo de una aplicación para web2py es la posibilidad de combinación de funcionalidades específicas (por ejemplo de herramientas técnicas y la publicación de notas e intercambio de datos como solución integrada en un solo servicio. Y el intercambio no quedaría relegado a la exhibición de información estadística ya que también sería posible la publicación de componentes instalables y módulos auxiliares.

Programada íntegramente en Python sobre el framework web2py [3]

Abierta a usuarios de todos los niveles (no se limita a aplicaciones para sistemas de instrucciones y notaciones específicas o sublenguajes técnicos).

Escenario preliminar: otras aplicaciones

Esta es una reseña del software con herramientas para matemáticas que preceden a math2py. No pretendo con esta breve lista hacer un estudio pormenorizado y completo de los recursos en software existentes ni comparar math2py con proyectos realmente avanzados, y de gran escala, como el caso de Matlab y SAGE. La idea es dar un simple panorama de las opciones con que se cuenta actualmente.

Matlab, de Mathworks:

MATLAB es "... un lenguaje de alto nivel y entorno interactivo que permite realizar tareas de cálculo intensivas con mayor velocidad que los lenguajes de programación tradicional como C, C++, y FORTRAN. ..." [4]. Se trata de una herramienta clásica y de amplia difusión (Inicialmente producida en 1984 [5]). Matlab es una aplicación propietaria (aunque es posible acceder a la interfaz de programación desde otros entornos).

A diferencia de Matlab, math2py no posee un lenguaje local para el ingreso de instrucciones, aunque en algunos casos se adoptó una sintaxis específica para permitir la carga de datos de entrada de las subaplicaciones. Por otra parte, para definición de funciones u operaciones es posible el ingreso de expresiones en Python o incluso JavaScript para funciones del lado del cliente.

Webmath, de Discovery:

Se trata de una aplicación on-line que permite al usuario explorar soluciones a problemas de matemática en forma interactiva. El sitio procesa datos ingresados por el usuario asociados a un problema determinado y presenta un planteo de solución aplicando una colección de soluciones programadas del lado del servidor. El proyecto está orientado al uso escolar y pedagógico pero no como un entorno programable. Webmath es una aplicación propietaria para la web.

En math2py es posible el desarrollo de soluciones como las presentadas por Webmath, con la ventaja del uso de herramientas avanzadas (helpers, sistema de plantillas, autenticación, componentes y widgets) incluidas en web2py.

Sage y The Sage Notebook: (open source):

"...Sage es un sistema libre y open-source con licencia GPL. Combina la potencia de muchos paquetes open-source existentes en una interfaz unificada basada en el lenguaje Python. Misión: Crear una alternativa viable open-source a Magma, Maple, Mathematica y Matlab. ..." [6]

The Sage Notebook es la interfaz web para sage. Permite correr un servidor que da acceso a herramientas para la producción de hojas con operaciones matemáticas y presentación de gráficos. Es una aplicación implementada en varios servidores, con acceso al código fuente y disponible como instalación. Las aplicaciones en línea son de libre acceso, creando una cuenta de usuario. math2py, al igual que SAGE utiliza la librería mpmath para el manejo de funciones.

Sage no es una herramienta práctica para un usuario inexperto. (Requiere ingreso de instrucciones Python avanzadas y dominio de la interfaz de programación) y está diseñada para un uso profesional y de investigación. El diseño de math2py busca presentar una interfaz rápida y fácil de utilizar para las operaciones más básicas y que permita complicar el funcionamiento de la aplicación gradualmente según el problema planteado, agregando elementos a la interfaz web.

Funcionalidades desarrolladas

- Álgebra lineal

  Resolver sistema, matriz aleatoria, combinación de operaciones, almacenamiento en servidor de datos de funciones y ecuaciones (block de notas de E/S), CSV, render LaTex de sistemas (a través de plugin_wiki de web2py y servicios de Google API)

- **Números:**

  Identificar

- Otras herramientas

  Math panel: acceso a funciones de librerías incorporadas y documentación desde el navegador. Gráficos de tablas. Render LaTex de ecuaciones. Acceso a valores ingresados por el usuario almacenados en la base de datos. Conversión y almacenamiento en el sistema como CSV. Cálculo de funciones. Envío de comandos Python del lado del cliente.

Presentación de algunas herramientas

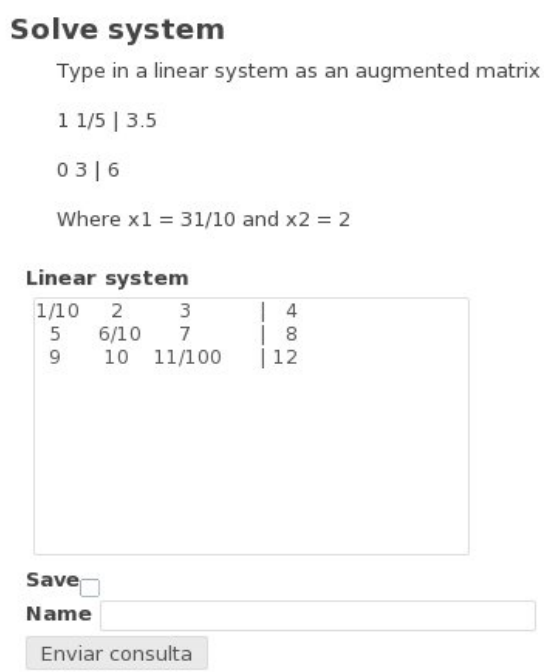
- Resolver sistema de ecuaciones

math2py resuelve sistemas de ecuaciones lineales por medio de la librería mpmath. La carga de los datos se hace utilizando la siguiente sintaxis:

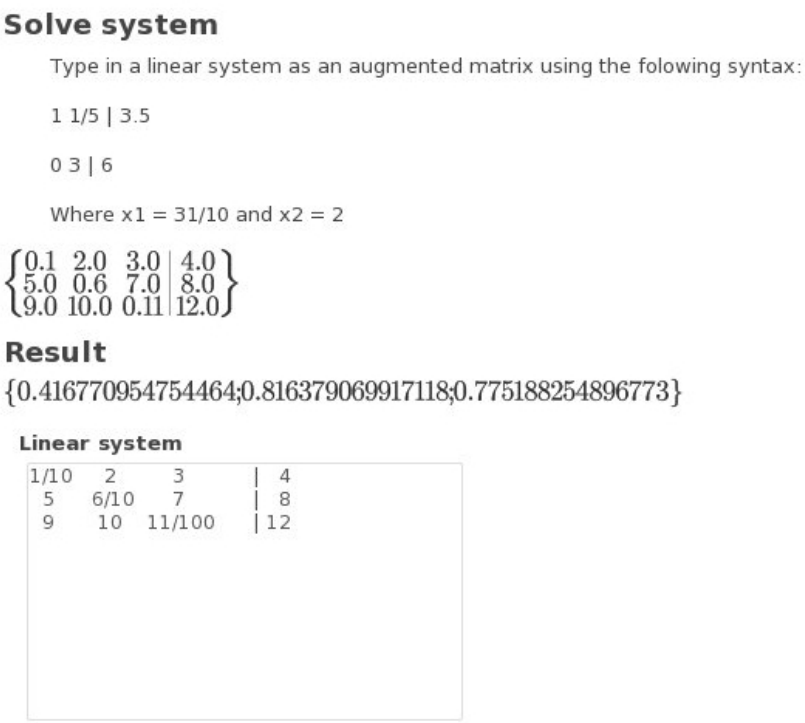
```
1    1/5    |    3.5
0     3     |     6
```

Para la solución $x_1 = 31/10$ y $x_2 = 2$

Ingreso de sistema



Solución



- Almacenamiento en servidor de operaciones

  La aplicación permite almacenar y recuperar sistemas lineales en memoria durante la sesión del navegador, a través del objeto session de web2py. Resolver el sistema almacenado: procesa un sistema de ecuaciones almacenado en memoria durante la sesión del navegador (último sistema guardado) y presenta la solución en pantalla.

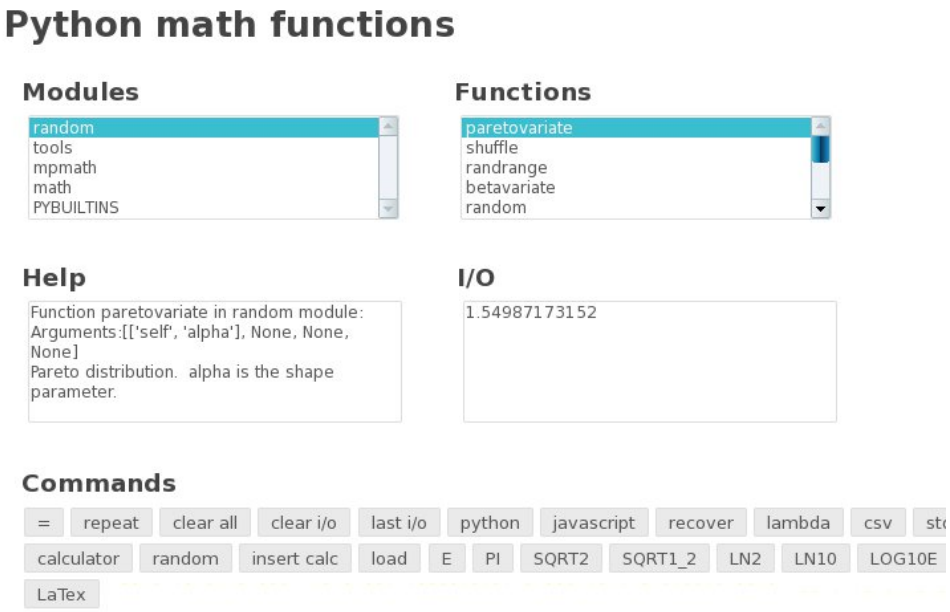
- Convertir y resolver sistemas de ecuaciones como valores separados por comas (csv)

  math2py puede convertir a y desde csv sistemas de ecuaciones procesando datos de tablas o de ingreso de texto del usuario en el cliente web.

- Otras herramientas:

Opciones/comandos para sistemas lineales. Matriz aumentada / valores aleatorios. Descomposición de Cholesky y vector aleatorio (combinación). Lista de matrices almacenadas.

Math panel



La función "=" procesa los argumentos en la caja de texto E/S para la función seleccionada. Argumentos de funciones: para el ingreso de texto (y presentación de resultados) se utiliza una caja de texo denominada I/O. La sintaxis (a, b, c, ...z) se adoptó para el ingreso de argumentos por similitud a la notación típica en lenguajes como Python o JavaScript. Ejemplo: (1,2,3), 4 es una lista compuesta de argumentos simples que contiene una secuencia (en la posición 1) seguida de un entero. La función de los paréntesis, a diferencia de su uso en Python, es diferenciar el inicio de una secuencia de argumentos de los argumentos simples de entrada.

La sintaxis para el ingreso de matrices es la siguiente (nótese la ausencia de comas):

```
[ [ x11 x12 ] [ x21 x22 ] ]
```

Esto equivale a la matriz:

```
|  x11  x12  |
|  x21  x22  |
```

Este tipo de notación es válido sólo para funciones que aceptan matrices como argumento. En el caso de los vectores columna la notación es:

```
[ [ x1 ] [ x2 ] ]
```

Que equivale a:

```
|  x1  |
|  x2  |
```

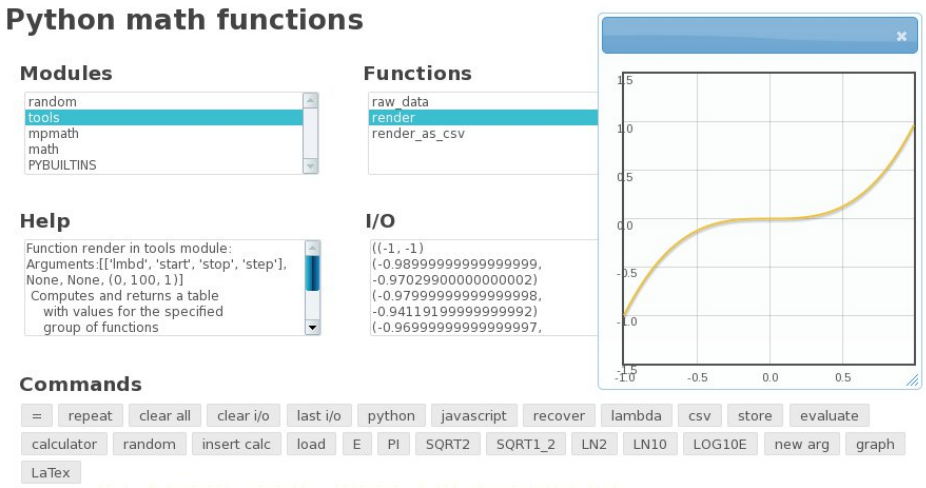
El segundo caso es el argumento b de matriz aumentada en mpmath y el primero el argumento A.

Se pueden embeber expresiones Python o código JavaScript en cada argumento por medio de llaves dobles {{ expresión Python }} o simples { código JavaScript } respectivamente.

Es válido el uso de n+nj para números complejos (sin paréntesis) en argumentos simples: 1-1j, (1+2j, 3.14+3.14j, 1). Estos tipos de valores son interpretados en el servidor. Como los valores numéricos de entrada son interpretados en el cliente web y procesados del lado del servidor con Python y JavaScript, la herramienta permite el uso de notación científica: 1e-10, 1.41e2+3.14j, (2.0005e-3, 0, 1)

La herramienta Math panel permite graficar funciones generadas por la función especial render, en el módulo auxiliar de math2py tools. Los parámetros de coordenadas para los gráficos son un conjunto de pares con la forma ((x1, y1), ... (xn yn)). math2py utiliza el servicio de generación de gráficos de la librería flot para jQuery. (La función del gráfico es $f(x) = x^3$)

Math panel: Gráficos



Funcionalidades pendientes

Agregado de librería de mpmath a la GUI web (solo se incluyó una parte mínima) Esquema de webservices para operaciones matemáticas inter-aplicación con JSON o SOAP [2]

Recursos

El proyecto está alojado en https://sourceforge.net/projects/math2py/

Descarga de aplicación

https://sourceforge.net/projects/math2py/files/

Código fuente

http://math2py.hg.sourceforge.net/hgweb/math2py/math2py/

Licencia

Affero AGPLv3 (con especificaciones sobre uso en sistemas en red)

Requerimientos/Instrucciones de instalación:

math2py corre en cualquier equipo que soporte el framework web2py.

La instalación es simple:

- Descargar el instalador para web2py desde el sitio del proyecto
- Carga del instalador con el comando Load en la interfaz administrativa de web2py.

En el caso de instalación como plugin, se recomienda tener como referencia la sección plugins del manual de web2py: http://www.latinuxpress.com/books/drafts/web2py/caps/capz13.html#plugins

Luego de instalar como plugin la aplicación puede cargarse cada componente por medio del helper LOAD o los atajos definidos en el modelo (El único atajo disponible actualmente es plugin_math2py(), que carga la herramienta Math panel).

Dependencias

- web2py
- plugin_wiki (plugin oficial de web2py)

[1] http://www.latinuxpress.com/books/drafts/web2py/caps/capz13.html#plugins "...Así que ¿ por qué es llamado plugin ? Porque proporciona un mecanismo para empacar un subconjunto de una aplicación y desempacarlo encima de otra aplicación (es decir, plug-in (conectar)). Bajo esta definición, cualquier archivo en su aplicación puede ser tratado como un plugin. ..."

[2] http://www.latinuxpress.com/books/drafts/web2py/caps/capz13.html#componentes "... Un componente puede estar compuesto de módulos, controladores y vistas, pero no hay ningún otro requisito estricto más que, cuando es incrustado en una página web, deba estar localizado dentro de una etiqueta html (por ejemplo un DIV, SPAN o IFRAME) y deba realizar su tarea independiente del resto de la página. ..."

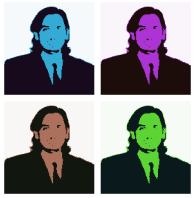
[3] Algunas funciones especiales fueron realizadas con jQuery/JavaScript para el desarrollo de la interfaz web.

[4] Traducción del sitio web en inglés.

[5] http://es.wikipedia.org/wiki/MATLAB

[6] Traducción del sitio web en inglés.

RAD2PY: plataforma Python para el Desarrollo Rápido de Aplicaciones bajo un Proceso de Software Personal



Autor: Mariano Reingart

Bio: Analista Programador y Docente. Entusiasta del Software libre Python, PostgreSQL y Web2Py en particular.

Web: http://reingart.blogspot.com

Email: reingart@gmail.com

Twitter: @reingart

En el campo del Desarrollo Rápido de Aplicaciones (método RAD) comúnmente existen diferentes aproximaciones metodológicas, teorías, estándares y buenas prácticas para llevar a cabo el Desarrollo de Software; generalmente sin una sólida fundamentación científica o sin basarse en datos empíricos completos e imparciales, agravándose sobre todo en la búsqueda de soluciones a las falencias o riesgos que presenta los enfoques ágiles.

A su vez, por estas dispersión conceptual, es difícil encontrar una herramienta (IDE) que de soporte integral al proceso, cuestión fundamental para la efectividad del método, que no puede ser salvada efectiva y eficazmente por los utilitarios independientes actualmente en existencia (editores, depuradores, gestores de proyectos, etc.), más allá de la calidad y capacidad de cada uno, dado que en este caso se confirmaría el axioma “el todo es más que la suma de sus partes”, debido a las propiedades emergentes de dichos sistemas complejos.

Por lo tanto, RAD2PY busca definir una base teórica y práctica para aumentar la productividad, resolviendo los problemas asociados al método RAD (aseguramiento de calidad, mejora continua); proveyendo una nueva generación de herramientas integradas guiada por el Proceso de Software Personal (PSP), para encauzar la metodología, solucionando sus falencias pero sin perder sus beneficios.

El Desarrollo Rápido de Aplicaciones

Ya en 1991, James Martin [RAD91] argumentó que los desarrolladores pueden archivar grandes reducciones en tiempo y costo, entregando sistemas de información de gran calidad al usar métodos modernos que combinan el involucramiento extensivo del usuario final con metodologías modernas de desarrollo soportadas por herramientas de diseño asistido por computadora (CASE) bien integradas (a lo que denomina I-CASE).

En relación a los riesgos y viabilidad del método, en un estudio de campo publicado por la ACM [RADRSK00], se destaca que si bien existe evidencia creciente que apoya a RAD como una mejora del “orden de magnitud” en la velocidad de construcción de software, emergen implicaciones serias. En primer lugar, es importante reconocer que RAD representa un cambio radical, por lo que se debiera realizar entrenamiento, no solo de conocimientos técnicos, sino ambién sobre las mejoras radicales que involucra. Mas importante aún, se deben evitar las capacidades RAD que deterioran las buenas prácticas en el desarrollo de sistemas, ya que solo se puede lograr alta calidad, bajo costo y desarrollo rápido si se utiliza una metodología de desarrollo (proceso de software disciplinado [RADSTN98]).

Es por ello que con RAD2PY se busca definir una base conceptual sólida, que permita desarrollar el método RAD de manera profesional, encuadrado con el Proceso de Software Personal (PSP) como lineamiento para definir y encauzar la metodología de desarrollo, para luego desarrollar una herramienta integrada (I-CASE) que le de soporte.

Si bien existen otros acercamientos al tema para solucionar los problemas de calidad, como ser el Método Dinámico de Desarrollo de Sistemas (DSDM) desarrollado a partir del RAD (SWQ01), que también se basa en enfocarse no solo en la velocidad sino también en la calidad. Sin embargo, a diferencia del proceso PSP más tradicional y robusto [MUKESH08], no existe evidencia empírica públicamente disponible y consensuada sobre la efectividad del DSDM [AGILDIR03].

El Proceso de Software Personal

El PSP fue desarrollado en 1993 por Watts Humphrey [PSP00], y es un marco personal y disciplinado de trabajo para llevar a cabo las tareas de software, que consiste en una serie de métodos, formularios y guiones que muestran a los ingenieros de software como planear, medir y administrar su trabajo. La meta del proceso es producir productos con cero defectos, respetando el calendario y con los costos planeados.

La efectividad de la metodología PSP fue documentada tanto en entornos académicos e industriales, ya sea en reportes y artículos técnicos. El resultado de estos estudios encontró que el PSP mejoró el rendimiento (estimación del tamaño y esfuerzo, calidad de producto y proceso) mientras que no tuvo desventajas en cuanto a la productividad

PSPEMP97.

Si bien estos estudios publicados demuestran las ventajas del PSP, diversos trabajos también señalan ciertos inconvenientes en la recolección manual de los datos y posterior análisis por parte de los desarrolladores, cuestionando su exactitud, recomendando que el uso de herramientas integradas debe ser necesario para un análisis de alta calidad de los datos del PSP PSPDAT98.

También se sugiere que si bien los estudiantes aprenden el PSP y logran mejorar sus procesos, lo abandonan cuando ya no se les requiere utilizarlo. Este "problema de adopción" puede ser producido por dos causas: sobrecarga de trabajo por la recolección y análisis métricas y el cambio de contexto (entre la herramienta de desarrollo y la recolección de métricas). PSPBEY03.

Actualmente existen tres generaciones de herramientas PSP:

Primera generación

el procedimiento original con guias y plantillas manuales con soporte de planillas de cálculo o bases de datos simples. Su principal inconveniente es la sobrecarga de trabajo y poca confiabilidad de los datos recolectados, barrera para la adopción del PSP.

Segunda generación

herramientas independientes para recolección de métricas y cálculo de estadísticas, citando como iniciativa destacada al Process Dashboard DASHBOARD11. Si bien solucionan parcialmente el problema de la sobrecarga de trabajo, introducen nuevos inconvenientes por el cambio de contexto (pérdida de concentración al alternar entre la herramienta de desarrollo y la herramienta de medición) y sigue dejando a criterio del programador el señalamiento de las fases (por ej. inicio y fin de codificación) y detección de incidentes (por ej. errores introducidos, errores solucionados), pudiendo compromenter la veracidad de los datos recolectados (ya sea por error u omisión).

Tercera generación

corresponden a marcos de trabajo para la medición y análisis dentro de procesos de ingeniería de software (in-process software engineering measurement and analysis - ISEMA), cuya implementación de referencia más destacada es Hackystat HACKYSTAT11 y consiste en aplicar discretamente telemetría (recolección automática de datos), que en teoría solucionaría la sobrecarga de trabajo, eliminando las distracciones por cambio de contexto y aumentando la confiabilidad de datos. Si bien hay casos que demuestran la factibilidad de usar las herramientas automatizadas AISEMA09, su uso no se a extendido debido a que introduce nuevos interrogantes relacionados a la privacidad de los desarrolladores y a la efectividad de los datos recolectados por ser e ocasiones demasiado voluminosos y de muy bajo nivel (genericos), dificultando su interpretación en tiempo y forma ISEMA07.

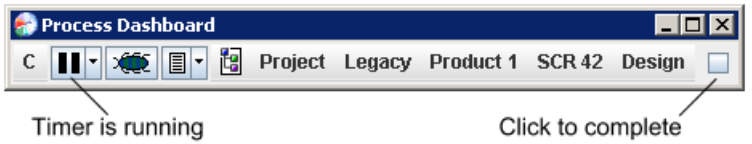


Figura 1: Ejemplo del Process Dashboard para la medición de tiempos y defectos

Como solución, para RAD2PY, se plantea integrar directamente la recolección de métricas dentro de las herramientas de desarrollo I-CASE del modelo RAD, solucionando de este modo los problemas antes planteados, generando una herramienta PSP de Cuarta Generación que integre completamente el proceso de desarrollo de software para simplificar y sustentar una efectiva disciplina personal, totalmente automatizada, sin necesidad de programas o tareas adicionales.

Vale aclarar que las herrramientas actuales estan desarrolladas en Java y son extremadamente complejas, lo que dificulta cualquier integración en Python.

Arquitectura de RAD2PY

Esta investigación se enfocara, en primer lugar, en desarrollar un Entorno de Desarrollo Integrado (IDE2PY), que de soporte al PSP (tanto a nivel fases como recolección de métricas). Para ello, se integrará la cadena de herramientas, rescatando los puntos sobresalientes de cada una:

- IDLE / PythonWin: entorno integrado, características de edición y depuración integradas.
- PyCrust: facilidades de consola interactiva
- PythonCard: construcción y modelado de GUI (utilizando wxPython)
- SPE: visualizador de objetos y clases UML
- PyFlakes: herramienta para detectar errores (similar a la compilación de los lenguajes estáticos).
- Py2Exe / Freeze: herramienta de despliegue

En segundo lugar, se desarrollará una herramienta de soporte y seguimiento (PSP2PY), preferentemente con interfase web, para la adquisición de requerimientos, seguimiento de errores, generación de documentación on-line; similar a las herramientas Trac,

RoundUp. Se tomará como referencia para la captura y análisis de métricas PSP, el sistema Process DashBoard. La integración de datos se realizará a través de una base de datos relacional.

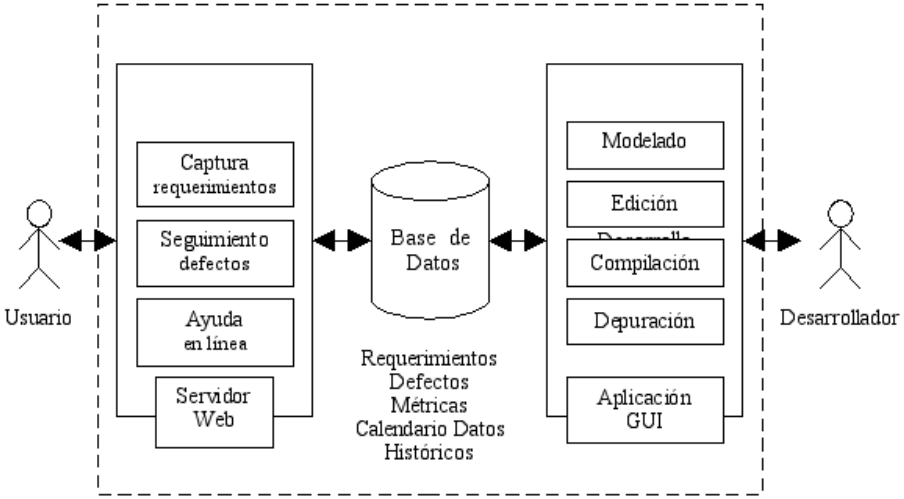


Figura 2: Esquema general de RAD2PY

WEB2PY es la herramientas más adecuada de introducir para obtener una mejor calidad desde el marco conceptual planteado, debido a que cumple todas las características de desarrollo rápido de aplicaciones (prototipado, iterativo, incremental, priorizando la funcionalidad y posee testeo integrado), es centrado en bases de datos relacionales y posee herramientas CASE totalmente integradas (IDE de desarrollo, pruebas, despliegue), y fue creado especificamente con las comunidades científica y académica en mente [WEB2PY11]

Del propio libro oficial [WEB2PY10] se extraen las siguientes características:

- web2py es una plataforma web libre de código abierto para el desarrollo ágil de aplicaciones web seguras, sustentadas en base de datos.
- web2py es una plataforma completa, lo que significa que contiene todos los componentes necesarios para construir aplicaciones web completamente funcionales;
- web2py está diseñado de manera que guía al desarrollador web a aplicar buenas prácticas de ingeniería de software, tales como el uso del patrón Model View Controler (MVC);
- web2py separa la representación de datos (el modelo) de la presentación de datos (la vista) y también de la lógica de la aplicación y flujo de trabajo (el controlador);
- web2py proporciona un sistema de tickets. Si ocurre un error, los tickets se expiden para el usuario, y el error se registra para el administrador.

Si bien WEB2PY es una herramienta simple e integrada que permite centrarse en la funcionalidad a desarrollar [PET10], dada la experiencia con la misma (y con el lenguaje Python en general), se considera necesario agregar varias funcionalidades para completar RAD2PY (segun las fases del PSP y enfoque previamente enunciado):

Planeamiento

- Soporte para registro de actividades/tareas, tiempos estimados y acumulados.
- Medición de tiempos automática (incluyendo interrupciones y detección de cada fase).
- Mejoras en el seguimiento a los tickets de defectos (introducidos y removidos en cada fase).
- Soporte para listas de comprobación (checklists) y guiones (scripts).
- Estimación según datos estadísticos.

Diseño

- Mejorar el soporte para documentación el proyecto (archivo de texto - formato wiki).

Codificación

- Facilitar las técnicas de revisión y detección temprana de defectos (“chequeo estático”).
- Medición automática de LOC (líneas código) eliminadas, modificadas, agregadas o reusadas.

Compilación

- Chequeo de sintaxis al editar los archivos -agregado-.
- Chequeo estático de nombres de variables y funciones definidas.
- Chequeo del estándar de codificación (convención de nombres, sangría, etc.).

Pruebas

- Depurador integrado.

- Generación de tickets en pruebas automatizadas.
- Mejora en tickets de error.

Al ser de código abierto y estar programado en Python hay gran cantidad de herrmientas libres que pueden ser utiles para el presente, y si bien se relevaron y analizaron muchas de ellas (ActiveGrid/PyIDE, wxPyDev, Pyragua, Picalo, SPE, Ninja-IDE, PythonWin, drPython, PythonCard, IDLE), ninguna era realmente integrada y simple, y finalmente se opto por un enfoque nuevo y minimalista para poder desarrollar los objetivos de la investigación.

Un punto importante fue definir el tipo de entorno (aplicación web o aplicación “visual” GUI). Inicialmente web2py es una aplicación web totalmente administrable por internet (por ej. con editor de código por el navegador). Esto es bastante útil para ambientes distribuidos, pero es bastante limitado para desarrollar una IDE integrada y simple, por lo que se evaluó la posibilidad de desarrollar un prototipo “visual” con interfaz gráfica de escritorio (IDE2PY y GUI2PY inspirados en facilidades existentes, como el editor predeterminado ”de facto” IDLE de Python, y herramientas de de desarrollo rápido “visuales” como PythonCard y su juego de herramientas gráficas wxPython).

Por esto último, no solo se contribuirá a la comunidad Python con un entorno de desarrollo totalmente integrado y simple guiado por sólidas prácticas de la ingeniería de software, sino que también posibilitará el desarrollo rápido de cualquier tipo de aplicación (ya sea de interfaz gráfica, web o de consola) de manera unificada.

Estado actual del Proyecto RAD2PY

El avance de la presente investigación no ha sido sin sobresaltos, encontrando dificultades en cada etapa del desarrollo, tanto por problematicas de las bibliotecas externas (wxPython, Mercurial, etc.) como también con módulos y bibliotecas estandar de Python (Shell y Depurador, DiffLib, Help, DocTests, etc.), ya sea por falencias en la documentación, funcionalidades no disponibles o comportamientos no esperados, entre otros.

Igualmente se ha podido avanzar y el estado prototipo actual del Entorno de Desarrollo Integrado (IDE2PY) es:

- El editor es funcional, con autocompletado de código y calltips básicos, incluyendo soporte para globales de web2py.
- Consola interactiva, con redirección segura de entrada/salida estándar.
- Depurador simple con soporte de interrupciones e inspección rápida.
- Soporte para repositorios Mercurial limitado (operaciones locales).
- Comparador visual de diferencias experimental.
- Ayuda integrada muy básica.
- Recolección automática de defectos (chequeo estático y pruebas documentadas)
- Medición automática de tiempos en cada fase de desarrollo, contemplando interrupciones.

La aplicación web de soporte (PSP2PY) posee en funcionamiento el modelo básico para gestionar proyectos PSP (principalmente el almacenamiento de datos historicos de defectos y tiempos), y se planea agregar interfaz de usuario imple y funcionalidades de planificación, estimación y seguimiento.

Para interconectar ambas aplicaciones, se ha desarrollado un cliente simple de comunicacion entre procesos utilizando JSON, aprovechando las características de web2py en este sentido para exponer servicios web.

Internamente, la aplicación IDE2PY almacena los datos de defectos y tiempos en bases de datos simples utilizando la persistencia de objetos Python nativa (Shelve y Pickle)

Si bien existe ciertas dificultades que pudieran causar inestabilidad de la herramienta, en lineas generales no se han encontrado mayores obstaculos que tornen inviable el desarrollo actual.

La gran combinación de sistemas operativos, versiones de python y biblitotecas relacionadas, sumado a las dificultades encontradas y falta de recursos, han sido factores para reconsiderar ciertos aspectos del alcance, tendiendo a simplificar y radicalizar aún más los objetivos y metas del proyecto.

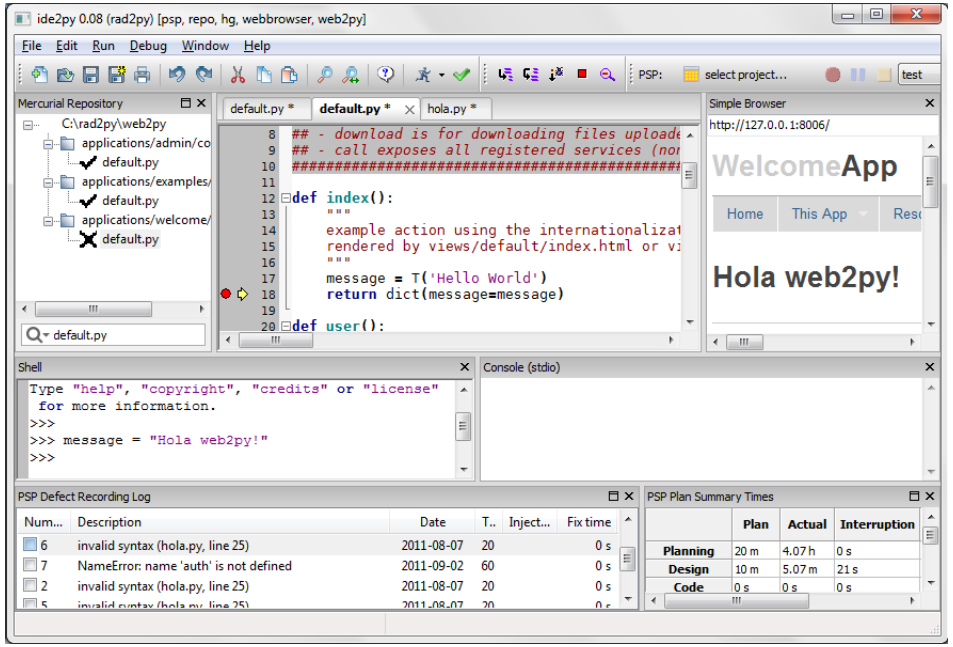


Figura 3: Captura de Pantalla de RAD2PY en funcionamiento (depurando una app web2py)

Implementación de RAD2PY

Internamente, el diseño de la IDE2PY es muy simple y modular, con un módulo principal (main.py), conteniendo la aplicacion en sí y la administración de la Interfaz Avanzada de Usuario (AUI) con la ventana madre, los menues, barra de herramientas y distintos paneles. A su vez, la aplicación se encarga de la configuración utilizando archivos de texto estandar .INI, con un archivo predeterminado (ide2py.ini.dist) que contiene la disposición básica de los paneles, editor y demás. De existir, esta configuración se mezcla con el archivo INI del usario que contiene la información particular sobre el usuario, parámetros recientes (archvivos abiertos), etc

La ventana principal (PyAUIFrame) puede ser extendida con mixins (Web2pyMixin, PSPMixin, RepoMixin) que funcionan a modo de clases complementarias heredándole comportamientos respecto a manejo de eventos y control de los elementos, pudiendo ser agregada en un futuro mayor funcionalidad sin modificaciones masivas.

El editor es un control basado en Scintilla (editor.py), con resaltado de sintaxis, autocompletado y calltips, y por cada archivo es creada una ventana separada que es manejada automáticamente con distintas solapas la Interfaz de Documentos Múltiples (AUI MDI). La introspección básica (con un espacio de nombres estático), se basa en evaluar/importar los modulos necesarios, por lo que en un futuro se buscarán métodos menos invasivos como PySmell. Para un correcto soporte de las codificaciones (UTF8, Latin-1, etc.) y formatos de archivos (BOM, saltos de linea unix, windows y mac) se desarrollaron funciones para manejo de archivos (fileutil.py)

El navegador (browser.py), implementado para probar los sitios web, es un control que deriva de webkit (gtk, en linux) o Internet Explorer (windows). Por el momento es bastante básico pero se planea agregarles botones de navegación y herramientas adicionales.

El interprete interactivo (shell.py) se basa en los controles proporcionados por wxPython, con adaptaciones para integrarlo con la consola (console.py) y depurador (debugger.py). Este último esta basado en el propio de Python (bdb, similar a la implementación de IDLE), con ajustes e integración con el editor. Se decidió tener la consola separada del interprete tanto por una cuestión visual como también para no afectar la interactividad del usuario. Si bien la consola puede redirigir la entrada y salida de procesos externos, por simplicidad en esta etapa de desarollo, el depurador y el interprete trabajan ejecutando los programas dentro del mismo proceso (similar a la implementación de IDLE y PythonWin). En un futuro se podría agregar un depurador remoto multihilo, además de otras mejoras visuales como un inspector de espacios de nombre y una ventana con la pila de llamadas.

El manejo de repositorios esta abstraido en dos capas, la primera de alto nivel (repo.py) con las operaciones básicas comunes a todos los repositorios, y la segunda especifica de cada sistema de versionado (repo_hg.py), en este caso con soporte para Mercurial, pudiendo agregarse otros sistemas. Existen eventos para la detección automática de cambios, y un arbol básico de los archivos del proyecto para buscar y operar sobre los mismos (agregar, borrar, comparar diferencias, comprometer, actualizar, etc.). Para comparar las diferencias se mejoro wxpydiff.py y se adaptaron los módulos estándar de python con un comparador de secuencias -FancySequenceMatcher- para detectar correctamente los cambios en el código fuente (diffutil.py).

Para el chequeo estático se integró PyChequer y PyFlakes (checker.py) que analizan el código fuente y reportan los defectos encontrados, complementado con pruebas de documentación DocTests (tester.py) que ejecutan las funciones y comparan los resultados esperados según la documentación de las mismas.

Para el soporte de desarrollo de aplicaciones web se embebió un servidor (web2py.py), que, gracias al enfoque único del framework (ejecución de los módulos en vez de importar permanentemente) permitio utilizar el depurador simplificado y habilitó la modificación del código sin necesidad de procesos externos o reinicios, y a la vez, el

concreto espacio de nombres global fue útil también para el editor (calltips y autocomplete). El webserver incorporado esta basado en la implementación estándar WSGI de Python, y es ejecutado en los momentos ociosos del loop principal de la aplicación, por lo que se pueden depurar aplicaciones de desarrollo pero no es recomendable para producción (o utilizarlo para aplicaciones dentro de la propia herramienta, como PSP2PY), ya que se pueden producir bloqueos.

El soporte al Proceso de Software Personal (psp.py) consiste en un listado de defectos y un grilla de tiempos en cada fase. Ambos son controlados por una barra de herramientas que permite iniciar, pausar y detener los cronómetros, elegir el proyecto y fase e ingresar defectos manualmente. Estos datos son almacenados en archivos persistentes de python (shelve/pickle) y luego son sincronizados con una aplicación web (en un servidor independiente) de soporte PSP2PY, utilizando procedimientos remotos con notación javascript (simplejsonrpc.py). De este modo se preserva la privacidad del desarrollador, eligiendo que datos y en que momento se envían, totalizandolos y pudiendo revisarlos y corregirlos.

Para los archivos de documentación, se esta implementando un editor de marcado simple (markmin) con previsualización (wiki.py).

Para el diseño de interfaces visuales (GUI2PY), se esta trabajando un editor HTML que dibuja los controles utilizando un diseño fluido wxHTML. El funcionamiento general será similar a una aplicación web pero reemplazando los artefactos HTML por controles nativos wxPython, y utilizando Python en lugar de Javascript, unificando y simplificando el diseño y disposición de interfaces y aprovechando las técnicas MVC y ayudantes del framework web.

Para más información sobre el avance, el software del proyecto se publica en [RAD2PY].

Desarrollo de Experimentos

Para la validación de las teorías, técnicas y herramientas individuales que sustenten el modelo elegido, se tomarán como muestras los datos experimentales recolectados por el prototipo para su posterior analisis conforme a los métodos estadísticos y matemáticos presentados en el Proceso de Software Personal [HUMPHREY95] para medir el rendimiento y progreso esperado logrado aplicando el modelo conceptual de la presente investigación.

Inicialmente, la selección de muestras será relacionada al desarrollo de los programas de ejemplos en los cursos del PSP, y luego de ser posible, al desarrollo de módulos de tres aplicaciones previas:

- Sistema de Facturación Electrónica, (1KLOC) [FACTURALIBRE]
- Sistema de Gestión de Emergencias, 911 (10KLOC) [AMPATU911]
- Sistema de Gestión Comercial, (750KLOC) [GESTIONLIBRE]

Observaciones ha destacar:

- Ya se ha comenzado con dicho desarrollo en Python (web2py) y no se han encontrado mayores inconvenientes,
- El desarrollo consiste en la migración de sistemas existentes, por lo que esta totalmente controlado al no esperar mayores desviaciones en tiempos de análisis y diseño, ya que se cuenta con el código fuente y documentación técnica para acelerar dichas etapas.
- La investigación se enfocará sólo en un grupo de módulos de ambos programas llevado a cabo por un único individuo, ya que el desarrollo en su totalidad excede el alcance y recursos disponibles de esta investigación.

No obstante, al ser proyectos de Software libre de Código Abierto, en un futuro es posible extender la investigación inicial sobre los mismos parámetros, con la colaboración de otros desarrolladores y entidades, para poder confirmar la hipótesis en una muestra mucho mayor.

Conclusión Preeliminar

Dado el supuesto central de la investigación, y motivo fundamental del proyecto RAD2PY, que consiste en que es posible el “Desarrollo Rápido de Aplicaciones” eficaz y eficientemente, para un profesional de manera independiente (proceso personal), sobre el marco conceptual y modelo planteado, se extrae del presente y sería posible conlcluir que, en principio, según lo investigado y experimentado hasta el momento (observando el avance del proptotipo inicial), las teorías, técnicas y herramientas han demostrado su viabilidad tanto en el plano teórico como práctico.

Luego de concluir el desarrollo del prototipo sería posible llevar a cabo los experimentos necesarios para recolectar los datos empíricos para confirmar la hipótesis planteada, que deberían traducirse en una mejora tanto en la productividad como en la calidad del trabajo de los desarrolladores de software.

Para finalizar, es menester mencionar un artículo en memoria de Watts Humphrey [BLOG@CACM10]_, que resume la importante relevancia de estos temas planteados, aduciendo que el trabajo de dicho autor será recordado por su búsqueda de que personas, equipos, proyectos, empresas y la industria del software en su conjunto apliquen los principios sólidos de ingeniería. A su vez, se afirma que no hay contradicción entre la práctica de metodologías ágiles y el enfoque disciplinado propuesto por el PSP (al menos en las no extravagantes, como el desarrollo iterativo), pero, sin embargo, también se comenta que la aplicación de dicho enfoque ha sido muy

específica debido a que la mayoría de los documentos de CMMI han sido escritos de forma demasiado burocrática, alejandolos de muchos programadores y gerentes que no pueden beneficiarse de dichos conceptos, pero que el PSP y otros principios seminales serán enseñados y practicados incrementalmente como parte de la inevitable profesionalización de la ingeniería en software.

Referencias

AGILDIR03 Abrahamsson, P., Warsta, J., Siponen, M. T., and Ronkainen, J. (2003). New directions on agile methods: a comparative analysis. In Proceedings of the 25th international Conference on Software Engineering (Portland, Oregon, May 03 - 10, 2003). International Conference on Software Engineering. IEEE Computer Society, Washington, DC, 244-254.

AISEMA09 Irina Diana Coman, Alberto Sillitti, Giancarlo Succi. Free University of Bozen-Bolzano, Italy. A case-study on using an Automated In-process Software Engineering Measurement and Analysis system in an industrial environment. Proceedings of the 31st International Conference on Software Engineering IEEE Computer Society Washington DC, USA ©2009 ISBN: 978-1-4244-3453-4; http://www.inf.unibz.it/~gsucci/publications/images/ACase-studyonUsinganAutomated In-processSoftwareEngineering.pdf

AMPATU911 Reingart Mariano. Bravo Angel, Policia de la Provincia de Buenos Aires. “AMPATU”: Proyecto de Gestión de Eventos de Emergencias 911. Alojado en Google Code.; http://ampatu.googlecode.com/

DASHBOARD11 The Software Process Dashboard Initiative; http://www.processdash.com/

FACTURALIBRE Reingart Mariano, Marcelo Alaniz, Alan Etkin, et al. Proyecto “Factura Electrónica Libre”: Interfases, herramientas y aplicativos para Servicios Web AFIP (Factura Electrónica) en Python. Alojado en Google Code.; http://pyafipws.googlecode.com/

GESTIONLIBRE Reingart Mariano et al. Proyecto “Gestion Libre” : Sistema de Gestión Administrativa y Contable. Alojado en Google Code.; http://ampatu.googlecode.com/

HACKYSTAT11 Hackystat open source framework project; http://code.google.com/p/hackystat

HUMPHREY95 Humphrey, Watts S. A Discipline for Software Engineering. Reading, MA: Addison-Wesley, 1995.

ISEMA07 Philip M. Johnson. Requirement and Design Trade-offs in Hackystat: An in-process software engineering measurement and analysis system. Proceedings of the 2007 International Symposium on Empirical Software Engineering and Measurement, Madrid, Spain, September, 2007.; http://csdl.ics.hawaii.edu/techreports/06-06/06-06.pdf

MUKESH08 Mukesh Jain. Delivering Successful Projects with TSP and Six Sigma: A Practical Guide to Implementing Team Software Process. Auerbach Publications, November 2008. ISBN 978-1420061437

PET10 Mariano Reingart. web2py para todos. PET: Python Entre Todos. La revista de la comunidad Python Argentina. San Isidro, Argentina. Agosto 2010. ISSN: 1853-2071; http://revista.python.org.ar/1/html/web2py.html

PROCEEDINGSTSP10 2010 TSP Symposium Proceedings Document. Carnegie Mellon University. Software Engineering Institute; http://www.sei.cmu.edu/tspsymposium/past-proceedings/2010/2010_TSP_Proceedings.pdf

PSP00 Watts S. Humphrey (2000) The Personal Software Process; Software Engineering Institute, Carnegie Mellon University

PSPA05 Raymund Sison, David Diaz, Eliska Lam, Dennis Navarro, Jessica Navarro. Personal Software Process (PSP) Assistant. In Proceedings of APSEC’2005. pp.687~696; http://www.csie.ntut.edu.tw/labsdtl/95-summer/0823-1.pdf

PSPBEY03 Johnson, P. M. et al (2003) Beyond the Personal Software Process: metrics collection and analysis for the differently disciplined. In Proceedings of the 25th international Conference on Software Engineering (Portland, Oregon, May 03 - 10, 2003). International Conference on Software Engineering. IEEE Computer Society, Washington, DC, 641-646.

PSPBOK09 PSP BOK Official Release.The Personal Software Process (PSP) Body of Knowledge, Version 2.0. Special Report. August 2009 (Revised February 2010). CMU/SEI-2009-SR-018; http://www.sei.cmu.edu/library/abstracts/reports/09sr018.cfm

PSPDAT98 Disney, A. M. and Johnson, P. M. (1998) Investigating data quality problems in the PSP. In Proceedings of the 6th ACM SIGSOFT international Symposium on Foundations of Software Engineering (Lake Buena Vista, Florida, United States, November 01 - 05, 1998). SIGSOFT '98/FSE-6. ACM Press, New York, NY, 143-152

PSPEMP97 Will Hayes, James W. Over (1997) The Personal Software Process: An Empirical Study of Impact of PSP on Individual Engineers, Technical Report; Software Engineering Institute, Carnegie Mellon University

RAD2PY Reingart Mariano. Proyecto "RAD2PY" : Rapid Aplication Development platform for python. Alojado en Google Code; http://code.google.com/p/rad2py

RAD91 James Martin, (1991) Rapid Application Development; Macmillan Publishing Co., Inc.

RADRSK00 Agarwal, R., et al (2000). Risks of rapid application development. Communications of the ACM 43, 11es (Nov. 2000)

RADSTN98 Stephen E. Cross (1998) Toward Disciplined Rapid Application Development, Department of Defense Software TechNews; Volume 2 Number 1 - Rapid Application Development (RAD) issue; http://www.dacs.dtic.mil/awareness/newsletters/technews2-1/toc.html

ROIPSP Rico, David F., What is the Return on Investment (ROI) of PSPSM (página web) http://davidfrico.com/roi-psppdf.htm

ROISPI04 Rico, David F. (2004) ROI of Software Process Improvement: Metrics for Project Managers and Software Engineers; J. Ross Publishing; http://davidfrico.com/

ROISTN02 Rico, David F. (2002) How to Estimate ROI for Inspections, PSPsm, TSPsm, SW-CMM®, ISO 9000, and CMMIsm, Department of Defense Software TechNews; Volume 5 Number 4 - Return-On-Investment from Software Process Improvement; http://www.dacs.dtic.mil/awareness/newsletters/stn5-4/

SWQ01 Baltus, B et.al. , Software Quality: State of the Art in Management, Testing, and Tools; Springer

WEB2PY10 Massimo Di Pierro, School of Computing, DePaul University. Web2py Enterprise Web Framework, 3rd Edition. Lulu.com. October 2010. ISBN 978-0557604142; http://www.web2py.com/book

WEB2PY11 Massimo Di Pierro, "web2py for Scientific Applications," Computing in Science and Engineering, vol. 13, no. 2, pp. 64-69, Mar./Apr. 2011, doi:10.1109/MCSE.2010.97

Nota: las páginas web fueron visitadas en el período de Junio de 2011

Vida y obra de objetos persistidos en ZODB

Autor: Orfi Schleppi

Bio: Estudiante de Ingenieria en Informática. Bajista de Random. Entusiasta del cine, el anime, las series de tv, el software libre y python :)

Twitter: @Orfx

ZODB (Zope Object Database) es una base de datos orientada a objetos para el almacenamiento transparente y persistencia de objetos python. Se incluye como parte de Zope (framework y servidor de aplicaciones open source escrito en python), pero también puede ser utilizado independientemente.

Persistiendo Objetos

Al instalar ZODB, hace de python un lenguaje persistente, permitiendo facilidades como la escritura automática de los objetos en el disco y la lectura cuando son requeridos por algún programa en ejecución.

ZODB utiliza el módulo internamente pickle, para persistir objetos. A continuación un ejemplo de como persistir un objeto pickeable:

```
class Conferencia:
    def __init__(self, nombre, anio):
        self.nombre = nombre
        self.anio = anio
    def titulo(self):
        return self.nombre.capitalize()
```

Escribimos algo

```
from  pickle import dump
pyconar = Conferencia ('PyconAr', 2011)
print pyconar.titulo(), pyconar.anio
f = open('data.pck', 'wb')
pycon = dump(pyconar, f)
f.close()
```

Lo leemos

```
from pickle import load
f = open('data.pck', 'rb')
pyconar = load(f)
print pyconar.titulo(), pyconar.anio #Pyconar 2011
print pyconar.__class__ **<Conferencia ...>
```

Y lo modificamos

```
from pickle import load, dump
f = open('data.pck', 'rb')
pyconar = load(f)
pyconar.anio = 2012
f.close()
out = open('data.pck', 'wb')
dump(pyconar, out)
out.close()
```

Entonces ahora podriamos preguntarnos: ¿Por qué no usamos “pickle“ para persistir objetos en lugar de complicarnos?

Porque eso podria llevarnos a perder “the big picture” ya que ZODB tiene muchas mas caracteristicas... Por ejemplo: administra los objetos para facilitar el trabajo del programador, manteniendo los objetos en memoria RAM, escribiendolos en disco cuando se modifican y eliminándolos de la memoria cuando no se hayan utilizado en suficiente tiempo.

Ahora si: ZODB

Un aspecto que deben cubrir los objetos es que deben ser persistentes. Para que un objeto python pueda ser persistido utilizando ZODB debe ser un dato primitivo (int, char, boolean, string, etc) o un objeto que pertenece a una clase que hereda la subclase persistent.Persistent.

Ejemplo:

```
>>> import ZODB
>>> from ZODB import FileStorage
>>> from ZODB.DB import DB
>>> import transaction
>>> storage = FileStorage('data.fs')
>>> db = DB(storage)
>>> connection = db.open()
>>> root = connection.root()
>>> root['pyconar'] = pyconar
>>> transaction.commit()
>>> db.close()
```

¿Y Ahora que ventajas tenemos?

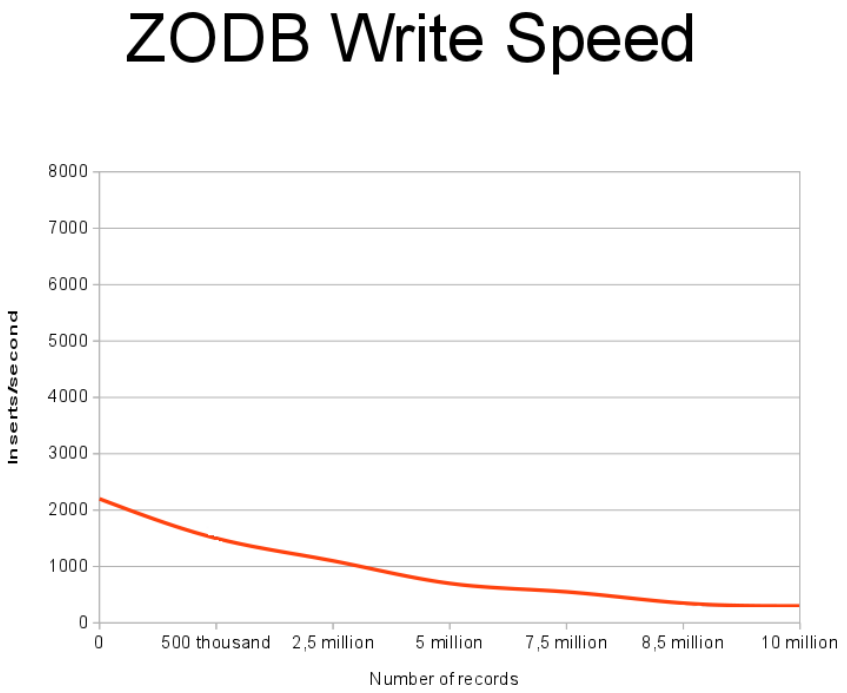
Lo primero es que la capa de mapeo a la base de datos desaparece, y pasamos de tener esto:

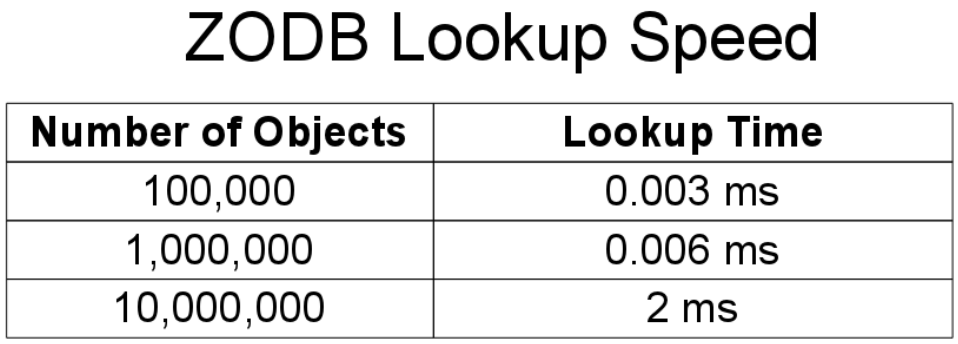
```
[ aplicación ]  <------> [ ORM ] <------->  [ BDD Relacional ]

   Objetos                                    Relacional / SQL
```

a esto:

```
[ apliación ]  <------>  [ ZODB ]

   Objetos               Objetos
```

Por otro lado los Benchmarks son muy alentadores, veamos este con PostgreSql





Number of Objects	Lookup Time
100,000	0.003 ms
1,000,000	0.006 ms
10,000,000	2 ms

Con estos datos podemos ver que:

- Los caches de gran tamaño son importante para escribir tanto como para leer.
- Necesitamos una solución para mas de 10 millones de objetos.

- ¿Si ZODB maneja 250 inserciones/segundos en el pico máximo de volumen de datos por que toma un minuto manejar 25 documentos?

Conclusiones

ZODB es una base de datos orientada a objetos bastante simple y transparente. No es una buena opción cuando se utilizan modelos de tablas.

Las consultas se hacen por interacción con objetos, no con la base de datos. Puede utilizarse para aplicaciones sensillas o escalables.

Para instalar

```
$ easy_install ZODB3
```

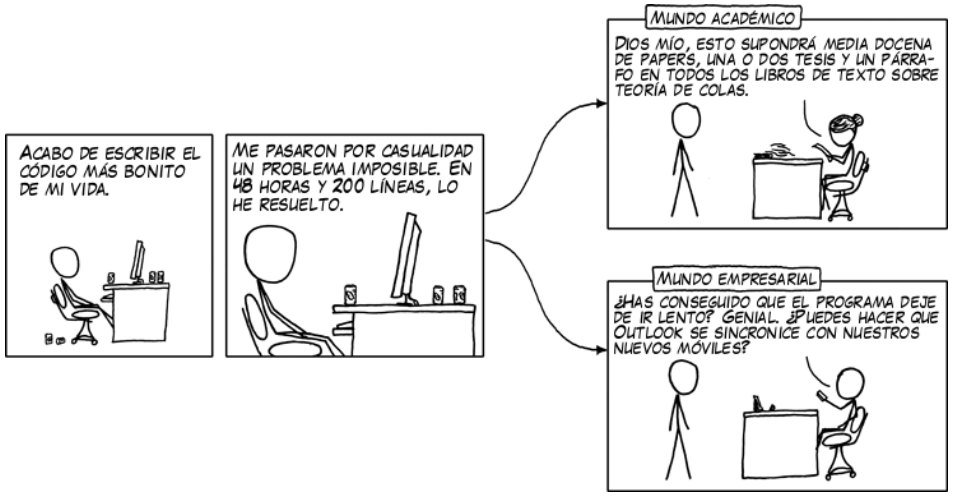
o

```
$ pip install ZODB3
```

Basado en:

- “ZODB: Apython Persistence System”, PyCon 2011, Chris Mc Donough (http://blip.tv/pycon-us-videos-2009-2010-2011/pycon-2011-zodb-a-python-persistence-system-4897373)
- “Tira tu base de datos relacional a la basura”, Roberto Allende (http://robertoallende.com/es/tira-tu-base-de-datos-relacional-a-la-basura)
- Proyecto: http://www.zodb.org/

xkcd



Academia vs. Negocios

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