Programación en Python

Patricio Páez Serrato

17 de Noviembre de 2002

Índice

1. Breve reseña.

Python es un lenguaje de alto nivel, interpretado, orientado a objetos, simple, extensible, libre, multiplataforma.

1.1. Python entre los demás lenguajes

Python es denominado lenguaje de 'scripting' en forma similar a Perl, con mayor funcionalidad que el shell. Será el languaje para scripting en Koffice. Elementos de C, Lisp, Modula-3.

www.python.org/doc/essays/comparisons.html

1.2. Historia de Python

Python fué desarrollado por Guido Van Rossum en el centro de investigación en Matematicas CWI en Holanda. Guido se basó en su trabajo anterior con el lenguaje abc. El nombre viene de un programa de la BBC 'Monty Python's flying circus'.

1.3. Principales características

Python toma las mejores características de varios lenguajes en una sintaxis sencilla, elegante, sin ambigüedades. Aporta la indentación como delimitador de bloques.

Los programas en Python suelen ser más cortos que en otro lenguaje por sus tipos de datos de alto nivel, la indentación y que no son necesarias las declaraciones de variables ni argumentos de funciones. A pesar de esto último, tiene más verificación de errores que C.

Python es compatible hacia atrás, los programas hechos en versiones anteriores funcionan en las versiones nuevas.

1.4. Disponibilidad del lenguaje en sistemas operativos

Python funciona en Gnu/Linux, Windows, Solaris, Sistema de la MacIntosh.

2. Cómo empezar

Puedes usarlo si tienes una PC con Windows o Gnu/Linux, una MacIntosh, etc. La versión actual es 2.2. Las anteriores son 2.1, 2.0 y 1.5.2.

2.1. Conseguirlo

- Windows: en www.python.org/2.2.1/python-2.2.1.exe
- Linux: seguramente viene con tu distribución ya compilado en tres paquetes RPM: python, python-docs, python-devel. Lo puedes también bajar de www.python.org/2.2.1
- Linux debian: mismos nombres de paquetes en www.debian.org/distrib/packages
- MacIntosh OS9, OS X: www.cwi.nl/~jack/macpython.html

2.2. Instalarlo

- Windows: correr el archivo ejecutable python-2.2.1.exe. Instala Tcl/Tk como lo ofrece el instalador, para poder usar IDLE y Tkinter, aunque no utilices Tcl.
- Linux RPM: rpm -i /mnt/cdrom/Mandrake/RPMS/python* o usa el instalador de sofware más amigable de tu distribución.

- Linux Debian: apt-get install python*.

2.3. Los manuales

Son cinco los principales:

1. Tutorial
2. Referencia del lenguaje
3. Referencia de la biblioteca
4. Extendiendo e incrustando
5. Instalando módulos de Python (administradores)

Vienen en formato html en el paquete python-docs, y van incluídos en el instalador de Windows.

2.3.1. Recomendaciones:

- Linux: crea un Bookmark en tu navegador preferido a /usr/share/doc/python-docs-2.1.1/index.html.
- Windows: los manuales quedan instalados en Start, Programs, Python 2.x, Manuals.
- Para imprimir: baja de www.python.org/doc/... los archivos pdf de cada manual. Te recomiendo que los lleves a engargolar con portada transparente.
- En Español: pyspanishdoc.sourceforge.net versión 2.0, y lucas.hispalinux.es/tutoriales/python/tutorial-python/tut.html versión 1.5.2.

2.4. Más información

Dependiendo de tu mejor forma de aprender, tienes los siguientes recursos para seleccionar:

2.4.1. Ejemplos

En el manual del Tutorial de Guido, vienen explicados.
En la Referencia de la Biblioteca, hay ejemplos para varios módulos.
Adam Kuchling tiene algunos ejemplos en www.amk.ca/python.

2.4.2. Libros

Existen ya varios en O'Reilly www.or.com y los puedes consultar en internet antes de comprar. También hay en www.amazon.com. Hay desde referencias de bolsillo hasta introducciones a la programación usando Python llenos de ejemplos.

2.4.3. Tutoriales

- El Tutorial de Guido Van Rossum
- www.python.org/doc/Newbies.html
- www.google.com python+tutorial

2.5. Sitios y grupos en dónde buscar

Usenet: comp.lang.python
web: www.python.org/doc/howto
google.com python+example

3. Interfases al usuario de Python

3.1. Ambiente interactivo

- Linux o Windows: teclea python en una consola.
- Windows: Start, Programs, python, python (command prompt).

python	entra a modo interactivo
python nombre-script	corre el script y termina
python -i nombre-script	corre el script y queda en modo interactivo
python -c 'comando' [argumentos]	ejecutar comandos de python y termina

Salir de modo interactivo: caracter EOF. Control-D Unix, Control-Z Windows.

Interrumpir programa: control-D.

- bash: #! /usr/bin/python en el primer renglón del programa para ejecutarlo como comando.

3.1.1. Teclas estilo emacs

c-A c-E, inicio/fin	Cursor a inicio/fin del renglón.
c-B c-F, cursor izquierda/derecha	Avanzar/retroceder un caracter.
c-K	Borrar de cursor hasta fin de renglón.
c-Y	Pegar últimos caracteres borrados.
c-_	Deshacer
_	Última expresión impresa

3.1.2. Historia de comandos

Cursor arriba, c-P	Comando anterior
Cursor abajo, c-N	Comando siguiente
c-R c-S	Buscar comandos hacia atrás/adelante

3.1.3. Autocompletar comandos

Crear archivo ~/.pythonrc conteniendo:
Import rlcompleter, readline
readline.parse_and_bind('esc: complete')

Antes de correr python: export PYTHONSTARTUP=~/.pythonrc

Más detalles en Apéndice A del Tutorial.

3.1.4. Parámetros de línea de comando

import sys
sys.argv lista de cadenas
sys.argv[0] es el nombre del script
sys.argv[1:] 1er. parámetro, 2do, etc.

3.2. IDLE

Integrated DeveLopment Environment para ambiente de ventanas.
Permite ejecutar paso a paso y muestra valores de variables.

- Linux consola: idle.
- KDE o Gnome: Applications, Development, Development environments, IDLE.
- Windows: Start, Programs, IDLE.

3.2.1. Teclas y atajos

alt-P	Comando anterior
alt-N	Comando siguiente
alt-/	Expandir palabra

Más info en menú Help.

3.3. Emacs

IDE (Integrated Developement Environment) poderoso, disponible en Windows, y con las siguientes facilidades:

- Sintaxis en colores

- Indicación de paréntesis
- Indentación automática
- Comentar una región
- Lista de funciones
- Ejecutar el buffer - no es necesario salvar
- Ejecutar el interpretador

3.4. Otros editores

Vim, activepython: www.activestate.com

3.5. Freeze

Para correr programas ya hechos en máquinas que no tienen python. Limitaciones.

4. Referencia del lenguaje

Para más detalles sobre:	consultar:
Comandos	Referencia del Lenguaje
Funciones y módulos	Refencia de la Biblioteca

4.1. Comentarios

Todo lo que esté a la derecha de #. Permitidos al final de líneas de continuación implícita.

4.2. Líneas lógicas y físicas

Línea lógica: una o mas líneas físicas, siguiendo las reglas de juntar líneas.

Explícita	'\' al final del renglón.
Implícita	(, [, y { pueden continuarse en siguiente línea física.

Varias instrucciones en el mismo renglón: separadas con ;.

4.3. Objetos básicos

Todo dato en python, incluso el código mismo, es un objeto. Los objetos tienen:

identidad	dirección en memoria, no cambia	id()
tipo	define qué operaciones y valores puede tomar	type()
valor	mutable, inmutable	

Algunos objetos son contenedores, es decir, tiene referencias a otros objetos.
Los tipos básicos son:

- Números. Valor único inmutable
- Secuencias. Contenedor. Conjunto ordenado, índices son números naturales. Mutables e inmutables.
- Mapeos. Contenedor. Conjunto sin orden, índices pueden ser un conjunto arbitrario excepto listas o diccionarios. Mutable.
- Objetos llamables. Funciones internas o definidas por el usuario, métodos, módulos internos o definidos por el usuario, clases, instancias de una clase, archivos.

Números	inmutables	enteros sencillos enteros largos punto flotante complejos	-2,147,483,648 a 2,147,483,647 sin límite doble precisión pareja de números de punto flotante	OverflowError z.real, z.imag
Secuencias	 inmutables mutables	 cadenas tuples listas	 ” ’a’ “hola“ ““““hola ““““ ( ), (1,), (1,2) [ ], [1], [1,2,3]	
Mapeos	mutables	diccionarios dbm, gdbm	{ },{a:123, b:456}	

Método: función que pertenece a un objeto.
Objeto.*método*
Otroobjeto.*método*

4.3.1. Valores lógicos

Falso	Valor cero, contenedor vacío, None
Verdadero	Todo lo demás, Ellipsis.

4.3.2. Funciones

cmp(x, y)	0 si x=y, 1 si x>y, -1 si x<y

4.4. Números

Enteros	decimal octal hexadecimal	234890 0766 0x0fee
Enteros largos		1L
Punto flotante		1.5, 1e100
Complejos		(5, 7j)

4.4.1. Funciones

int(x, [base])	str o int ->int trunca
long(x)	str o int ->long
float(x)	str o int ->float
hex(i)	int ->str
oct(x)	int ->str
round(x, n)	float ->float, n=0
trunc(x)	xxx ->xxx
abs(x)	valor absoluto

4.5. Secuencias

inmutables	cadenas	'' 'a' 'ab'
	tuplas	() (1,) (1,2)
mutables	listas	[] [0] [0,1]

4.5.1. Elementos

nombre[índice]

nombre[0]	primer elemento
nombre[1]	segundo elemento
nombre[-1]	último elemento
nombre[-2]	penúltimo elemento

Índice fuera de rango es un error.

4.5.2. Secciones

nombre[índiceinferior : índicesuperior]
Longitud del corte = índicesuperior - índiceinferior

nombre[:fin]	Primer índice omitido vale 0.
nombre[inicio:]	Segundo índice omitido vale tamaño del objeto.
nombre[:]	Copia del objeto.
nombre[0:1]	Primer elemento.
nombre[0:2]	Primero y segundo elementos
nombre[-2:]	Últimos dos elementos

Índices fuera de rango se aceptan: topan con 0 y el tamaño del objeto
nombre[:i] + nombre[i:] es igual a nombre.

4.5.3. Funciones

max(s	Mayor elementos de s
min(s)	Menor elementos de s
len(s)	Cantidad de elementos

4.5.4. Operadores

+	Concatenación.
*	Multiplicación.
'aaa' 'bbb'	Concatenación implícita.
><==	Comparaciones - lexicográficamente.
e[not] in s	Membresía

4.5.5. Programación funcional

range([inicio,] fin [, incremento])	inicio=0, incremento=1[0, 1, ... incremento <fin] inicio+i*incremento <fin [inicio, inicio+i*incremento, ...]
xrange()	ídem, para rangos muy grandes
filter(función, secuencia)	if función(elemento): tomar elemento
map(función, secuencia [,secuencia...])	función(elemento [, elemento...])
reduce(función, secuencia)	función(elemento0, elemento1), etc.
zip(secuencia [,secuencia...])	Lista de tuplas con elemento i de cada secuencia)

4.6. Cadenas

''	Cadena vacía
'c'	Caracter
'abc'	Comillas sencillas
"abc"	Comillas dobles
'dice "hola"'	Comillas dobles dentro de sencillas
"buyer's guide"	Comilla sencilla dentro de dobles
'"buyer\'s" guide'	Comilla con escape
"""hola"""	Pueden contener varios renglones y comillas.
'''hola'''	ídem

4.6.1. Métodos

capitalize()	Iniciales con mayúsculas
s.join(secuencia)	Concatena secuencia con la cadena s
lower()	Minúsculas
lstrip()	Quita espacio en blanco al inicio
rstrip()	Quita espacio en blanco al final
splitlines([keepends])	Lista de renglones
strip()	Quita espacio en blanco
upper()	Mayúsculas

4.6.2. Caracteres

Son cadenas de longitud 1, no hay tipo char.

ord(c)	código ascii del caracter
chr(i)	caracter correspondiente
str(o)	cadena para imprimir representación del objeto
repr(o)	ídem, para usarse en eval()

4.6.3. Caracteres especiales

Secuencia de escape	Significado
\\	\
\'	'
\"	"
\f	formfeed
\n	linfeed
\r	carriage return
\t	tab
\ooo	caracter ascii ooo octal

Prefijo r son cadenas raw: no se eliminan los \.

4.7. Tuplas

tuple(secuencia)	tupla con los elementos de la secuencia

4.8. Listas

4.8.1. Métodos

append(x)	Agrega un elemento. Equivale a a[len(a):] = [x]
extend(L)	Extiende la lista agregando los elementos de la lista. Equivale a[len(a):] = L
insert(i, x)	Inserta un elemento en la posición dada.
insert(0, x)	Inserta al principio
remove(x)	Elimina el primer elemento cuyo valor es x. Error si no existe
pop([i])	Elimina el elemento en la posición dada, y regresa su valor. Sin parámetro es el último elemento.
index(x)	Regresa el índice del primer elemento con valor x. Error si no existe
count(x)	Regresa el número de veces que x aparece en la lista.
sort([cmpfunc])	Ordena los elementos, sobre la lista misma. cmpfunc(x y,) regresa 1 si x>y, 0 si x==y, -1 si x<y. *cmp*(x, y)
reverse()	Ordena los elementos al revés, sobre la lista misma.

4.8.2. Funciones y operadores

del a[n]	Borra elemento
del[x:y]	Borra sección
list(secuencia)	Lista con elementos de la secuencia

4.8.3. Usos

Pila LIFO	append(x), pop()
Cola FIFO	append(x), pop(0)
Matriz	L[r][c]
Lista recursiva	L.append(L)

4.9. Diccionarios

4.9.1. operaciones y métodos

len(a)	Cantidad de elementos en a
a[k]	Elemento de a cuya llave es k (subscripción)
a[k] = x	Asignar valor x a a[k]
del a[k]	Eliminar a[k] de a
a.clear()	Eliminar todos los elementos de a
a.copy()	Una copia de a
a.has_key(k)	1 si a tiene llave k, 0 de lo contrario
a.items()	Una copia de la lista de parejas (llave, valor) de a
a.keys()	Una copia de la lista de llaves de a
a.update(b)	for k, v in b.items(): a[k] = v
a.values()	Una copia de la lista de valores de a
a.get(k[, x])	a[k] si a.has_key(k), x de lo contrario

4.9.2. Código

eval(*expresión*)	Evalúa expresión
exec(*comando*)	Ejecuta instruccion(es)

4.10. Asignaciones

nombre = *expresión*
a,b = c,d (pack, unpack)
Las asignaciones no copian datos, ligan nombres a objetos. Siempre van al ámbito interno.

4.11. Operadores

	máxima prioridad	
llamada a función	f(argumentos)	
sección	x[índice:índice]	
subscripción	x[índice]	
referencia a un atributo	x.atributo	
unitarios	+ - ~	
potencia	**	
aritméticos	* / % + -	
desplazar bits	>> <<	
a nivel bits	& ^ |	and xor or
comparaciones	<> == >= <= <> !=	Se pueden encadenar: x <y <z
identidad	is [not]	
membresía	[not] in	
booleanos	not x and or	
lambda	expresión lambda	

4.12. Control de flujo

if expresión:
    instruccion(es)
[**elif**:
    instruccion(es)]
[**else**:
    instruccion(es)]

while expresión:
    instruccion(es)
[**else**:
    instruccion(es)]

for elemento in lista:
    instruccion(es)

continue	siguiente ciclo
break	salir del ciclo, no se ejecuta else
pass	no hace nada, sigue

4.13. Entrada y Salida

4.13.1. Consola

print	Renglón en blanco
print x,y,z	Separados por un espacio
print alfa,	Evita línea nueva al final
print '*cadena-de-formato*' % (expresiones)	'%10s, %5d' como en printf()
print '*cadena-de-formato*' % diccionario	'%(llave)formato'
rawinput([*mensaje*])	Entrada de datos. Import readline opcional.

4.13.2. Columnas

import string
print string.**ljust**(*expr*, ancho), string.**center**(*expr*, ancho), string.**rjust**(*expr*, ancho)

4.13.3. archivos

f=**open**('ruta-a-archivo' [, modo])	modo: 'r', 'w', 'a', 'rb', 'wb', 'ab'
f.**read**([bytes])	bytes=todos
f.**readline**([bytes])	'\n' incluído al final de la cadena. eof es cuando regresa cadena vacía.
f.**readlines**([bytesaprox])	bytesaprox=todos
f.**write**(*expresión*)	
f.**writelines**(lista)	
f.**tell**()	
f.**seek**(offset, respectoa)	0 principio, 1 posición actual, 2 fin del archivo.
f.**close**()	

Predefinidos: sys.stdin sys.stdout sys.stderr

4.13.4. Objetos persistentes

```
import pickle
pickle.dump( objeto, archivo)
objeto = pickle.load( archivo)
```

4.14. Funciones

4.14.1. Definición

```
def nombre ( [parámetros] ):
    "Cadena de documentación."
    instruccion(es)
    [return [expresion(es)] ]
```

4.14.2. Parámetros

Posicionales	nombre, ...
Con valor por omisión	nombre=*valor_por_omisión*, ...
Opcionales	*nombre

4.14.3. Uso

nombre([*argumentos*])

4.14.4. Argumentos

Posicionales	valor, ...
Con nombre	nombre=*valor, ...*
Por tupla	**tupla*
Por diccionario	***diccionario*

Orden: de arriba a abajo.
Los argumentos pasan por referencia al objeto.

4.14.5. Funciones lambda

lambda *parámetros*: *expresión.* Solamente una instrucción.

4.14.6. Documentación

Empieza con mayúscula, termina con punto, no incluye el nombre del objeto. Segunda línea vacía si tiene más líneas.

4.15. Ámbitos y espacios de nombres

Espacio de nombres: mapeo de nombres a objetos. Implementado con diccionarios.

Alias: varios nombres pueden apuntar a un mismo objeto.

Ámbito: región de texto de un programa de python donde un nombre es accesible directamente (con referencia sin calificar). Determinados estáticamente, usados dinámicamente.

Siempre 3 ámbitos, se buscan en este orden:

Interno	Nombres locales de la función actual.	locals(), dir(), vars()
Intermedio	Nombres globales del módulo actual.	globals()
Externo	Nombres interconstruídos	

4.16. Clases

4.16.1. Definición

class nombre:
    instruccion(es) (usualmente definiciones de funciones)
[**def** __init__ (self):] se ejecuta en cada instanciación

inal de ejecutar la definición se crea un objeto clase, el nombre de la clase se liga al objeto clase en el espacio de nombres local.

4.16.2. Objetos clase

rencias de atributo nombreclase.*atributo*
Asignaciones de atributo nombreclase.*atributo* = *expresión*

4.16.3. Instanciación de clase

x = nombre([*argumentos-para-init*])

4.16.4. Objetos de instancia

Atributo: x.*atributo* - variables
Método: *x.metodo([argumentos])* - funciones

4.16.5. Objetos método

Primer argumento es el objeto al que pertenece: def metodo(self [,parametro(s)]):

4.16.6. Herencia

class *clasederivada*([modulo.]*clasebase* [, *clasebase2* ...]):
Pueden remplazar los métodos de la clase base.

4.17. Módulos

Módulo es un archivo que contiene instrucciones de pyhton y definiciones de funciones o clases.
Dado *módulo*.py:
import *módulo*
from *módulo* **import** *
reload *módulo*

4.17.1. Funciones

dir([módulo])	Nombres que define el módulo.
dir()	Nombres locales, equivale a dir(__main__)
import __builtin__ dir(__builtin__)	Nombres interconstruídos.
vars([módulo])	Diccionario de símbolos del módulo.

4.17.2. Módulos existentes

__main__	Instrucciones ejecutadas en el interpretador, con script o interactivo.
__builtin__	Funciones y variables intercontruídas.

4.17.3. Ruta de búsqueda de módulos

sys.path = directorio actual, PYTHONPATH, ruta de la instalación de Python.

4.17.4. Módulos compilados

Al importar por primera vez, el interpretador genera código de bytes y crea el archivo *módulo*.pyc.

Es posible correr con .pyc solamente.

4.17.5. Paquetes

Un directorio con subdirectorios y uno o más módulos.

Debe existir __init__.py en directorio principal y cada subdirectorio, puede estar vacío.

Variable __all__ indica cuáles módulos importar con **from** *paquete* **import** *.

import *paquete.módulo.submódulo*

from *paquete* **import** *módulo, ...*

4.17.6. Módulos externos

Revisar en news:comp.lang.python.announce

csv www.object-craft.com.au/projects/csv lectura de archivos exportados de Excel, etc.

pydoc.py genera documentación automáticamente.

inspect.py

4.17.7. Distutils

Python 2.0 en adelante:

[python *setup.py* build [–build-base=/*ruta*]]

python *setup.py* install

4.18. Excepciones

try:
    instruccion(es)
except [excepción(es):
    instruccion(es)
else:
    instruccion(es)
sys.exc_info() tupla con (tipo, valor, traza)
import exceptions; dir(exceptions)

5. Interfases gráficas de usuario (GUIs)

Principales opciones para tener interfase gráfica en una aplicación de python.

Toolkit gráfico	Módulo python	Gnu/Linux	Windows	Mac	Usado en
Tk	Tkinter	sí	sí	sí	Tcl
GTK	pygtk	sí	sí	sí	Gnome
Qt	pyQt	sí	sí	sí	KDE
wxWindows	wxpython	sí	sí	sí	Chandler

5.1. Terminología

Widgets: elementos de una interfase: botones, menús, ventana, marco, etc.
Ligas (bindings): interfase de programación de un toolkit a un lenguaje.
Jerarquía: relación de los widgets que forman una interfase.
Empacar: colocar un widget en otro contenedor.
Marco: Agrupador de otros widgets en diseños complejos. Clase base para implementar widgets compuestos.
Señales: acción del usuario con el teclado o el ratón.
Eventos: función que responde a una señal.
Manejadores (handlers): función que ejecuta un evento.

5.2. Tk

Tk incluído en distribuciones de Linux v 8.3..., y en el instalador de python para Windows.
Tkinter módulo Tkinter. Tix contiene widgets de mayor complejidad.

5.2.1. Referencia

An introduction to Tkinter. Fredrik Lundh. Muchos ejemplos.
www.pythonware.com/library/an-introduction-to-tkinter.html.

5.2.2. Ejemplos

pysol www.oberhumer.com/opensource/pysol

5.2.3. Estructura básica

```
from Tkinter import *
root = Tk()
algo = nombre-widget(root)
algo.pack()
root.mainloop()
```

nombre-widget: Menu, Scrollbar, Button, TopLevel, Frame, Canvas, Text, etc.

5.3. Gtk

Gimp Toolkit. www.gtk.org. www.gtk.org/api/FAQ y /tutorial
pygtk módulo pygtk www.daa.com.au/~james/pygtk
pygnome módulo gnome.ui www.daa.com.au/~james/gnome

5.3.1. Referencia

Uso de widgets desde pygtk, por James Henstridge: www.gnome.org/~james/pygtk-docs.

Se puede bajar pygtk-docs.tar.gz.

www.async.com.br/faq preguntas y respuestas.

5.3.2. Ejemplos

theopenlab.uml.edu/pygtools
/usr/share/doc/pygtk-0.6.8/exampes/... testgtk, neil, glade, imlib
/usr/share/doc/pygnome-1.4.1/examples/...

PyGTK tutorial, John Finlay. Excelente introducción a Gtk.
www.moeraki.com/pygtktutorial/pygtktutorial.tgz

Python Gnome Tutorial, Daniel Kornhauser. daniel@bq.unam.mx
laguna.fmedic.unam.mx/~daniel/pygtktutorial

5.4. Libglade

Permite que aplicaciones carguen la interfase de un archivo XML durante la ejecución. Automáticamente conecta manejadores a las señales definidas en el archivo XML. Usa gtk.

pygtk-libglade módulo libglade.

pygnome-libglade módulo libglade para widgets gnome.

5.4.1. Referencia

developer.gnome.org/doc/API/libglade/libglade.html (desde C)

5.5. Glade

Un generador visual de interfases gráficas para GTK. glade.gnome.org.

5.5.1. Referencia

Guía de usuario (muy completa), Preguntas más frecuentes (acerca de señales y manejadores), y Tutorial. En Help de Glade, y /usr/share/gnome/help/glade/c/.

5.5.2. Ventanas de primer nivel

Gtk	Gnome
GtkWindow	GnomeApp
GtkDialog	GnomeDialog
GtkFileSelection	GnomeMessageBox
GtkColorSelectionDialog	GnomeAbout
GtkFontSelectionDialog	GnomePropertyBox
GtkInputDialog	

5.5.3. Ejemplos

Glade y libglade en python. Parte 1.
Hilarie Fernandes hilaire@ofset.org
www.linuxfocus.org/English/July2000/article160.shtml bajar y renombrar:
color_glade.txt –>color.glade, couleur_py.txt –>couleur.py
Parte 2.
www.linuxfocus.org/English/January2001/article224.shtml ????

Pygnome, pygtk and libglade tutorial.
autor...
sjbrown.geeky.net/metagame-sector/tutorial.html. hello y simple.

Writing gnome applications with Glade and Python, Robert Laing.
www.icon.co.zq/~zapr/project1.html

5.6. Qt

www.trolltech.com. Es GPL desde Noviembre 2001.

5.7. wxWindows

www.wxwindows.org, www.wxpython.org

5.7.1. Ejemplos

Karel el robot pykarel.sourceforge.net