Interfaces gráficas en Python

TkInter

PyGTK

PyQT

WxPython

David López Fernández

I.T en Informática de Sistemas

1º Ingeniería Informática

Programación avanzada

Índice de contenido

1 TkInter

Introducción

Es un módulo que nos permite construir interfaces gráficas de usuario multiplataforma en Python utilizando el conocido toolkit Tk. Python incluye este módulo por defecto, lo que hace que sea un tookit muy popular. TkInter, además, es robusto, maduro y muy sencillo de aprender y de utilizar, contando con una amplia documentación.

TkInter se distribuye bajo la PSFL (Python Software Foundation License) una licencia compatible con la GPL creada para la distribución de software relacionado con el proyecto Python. La PSFL carece de la naturaleza viral de la GPL, por lo que permite crear trabajos derivados sin que estos se conviertan necesariamente en software libre.

Estos son los pasos a seguir para crear una aplicación en Python con el entorno ToolKit (Tk).

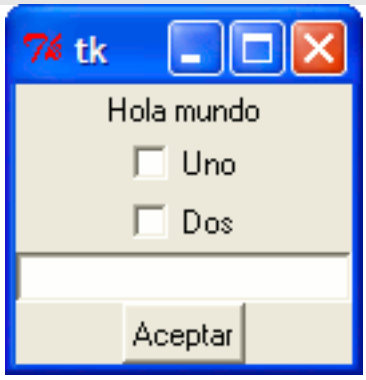
1. Importar las clases y constantes del módulo Tkinter, mediante un comando import en su forma
   ```
   import Tkinter
   ```
2. Crear una instancia de la ventana raíz, a partir de la clase *Tkinter.Tk()*.
3. Crear una instancia de uno o más elementos para insertar en esta ventana raíz. Los elementos pueden ser widgets *Label, Canvas, Frame*, y otros. Un widget procesador de texto por ejemplo, se crea a partir de la clase *Tkinter.Text()*.
4. Enlazar funciones o métodos de la ventana raíz o alguno de los widgets a uno o más eventos del usuario. Los eventos son los movimientos y acciones del ratón y teclas presionadas en el teclado.
5. Mostrar cada elemento con el método *pack(),* o estableciendo una rejilla*, grid()*.
6. Agregar información a los elementos, como puede ser texto al editor o a una etiqueta
7. Llamar el método *mainloop()* de nuestra ventana raíz, para que se active la atención a los eventos.
8. Para terminar la aplicación se llama al método *quit()* de la ventana raíz.

Esto es una pequeña aplicación escrita en Python+TkInter:

```
from Tkinter import *

root = Tk()
frame = Frame(root)


label = Label(frame, text="Hola mundo")
c1 = Checkbutton(frame, text="Uno")
c2 = Checkbutton(frame, text="Dos")
entry = Entry(frame)
```

```
button = Button(frame, text="Aceptar")
```

```
frame.pack()
label.pack()
c1.pack()
c2.pack()
entry.pack()
button.pack()

root.mainloop()
```



A continuación se profundizará un poco en el uso de los diferentes widgets que proporciona TK.

Widgets

Uso de Labels

Como siempre, en todos los lenguajes, la primera aplicación es hola mundo, y ahora se creará un label. Primero, se crea el formulario:

```
from Tkinter import *
root = Tk()
root.mainloop()
```

Se crea el label:

```
holamundo = Label(root, text="¡Hola Mundo!")
```

El declarador llamado *holamundo*, contiene un widget de caracter "LABEL", se usa root para que el programa identifique esa función como tkinter.

Para declarar este label en el form se utiliza pack() o grid(). Pack, hace un autoajuste y lo coloca donde el desea, pero con grid, se puede elegir columnas, lineas, situación...

```
holamundo.grid(row=1, column=1)
```

Queda así:

```
from Tkinter import *
root = Tk()
holamundo = Label(root, text="¡Hola Mundo!")
holamundo.grid(row=1, column=1)
root.mainloop()
```

Creación de Buttons

Los widgets buttons, sirven para efectuar una función. Esa función se declara antes con def(): y después se introduce una subfuncion en el button llamada command.

Lo primero es la declaración de sintaxis:

```
nombre = Button(root, texto del botón, comando de función,
tamaño de ancho y/o alto)
grid.nombre(linea, columna)
```

Primero, el texto del botón, es el "label" que saldra en un boton, por ejemplo si se quiere que el boton ponga "De acuerdo" escribiremos:

```
Button(... text="De acuerdo" …)
```

Ahora hay que crear la función:

```
def nombre de la funcion():
     código de la función
```

Ejemplo:

```
holamundolbl = Label(root, text="¡Hola Mundo!")
holamundolbl.grid(row=2, column=1)
```

Una vez se tiene el nombre de la función y el código, se pone en command el nombre de la función, ejemplo:

```
Button(...command=holamundo...)
```

Esto llamará a la función holamundo, y por último, se pone el width=20.

```
Button(...width=20)
```

El ejemplo queda como sigue:

```
from Tkinter import *
def holamundo():
     holamundolbl = Label(root, text="¡Hola Mundo!")
     holamundolbl.grid(row=2, column=1)
root = Tk()
holamundo      =      Button(root,      text="Activar      Hola
Mundo",command=holamundo, width=20)
holamundo.grid(row=1, column=1)
root.mainloop()
```

Utilizando TextBox's

Los textbox's en Python principalmente, se usaran con Strings, cadenas de texto. Antes que nada, hay que declarar el nombre del textbox como variable de string.

```
mi_variable_string = StringVar()
```

De esta manera, mi_variable_string que seria el nombre de la variable, identificada como String. Ahora, la declaración del textbox.

```
el_nombre_de_nuestro_textbox           =           Entry(root,
textvariable=mi_variable_string)
```

Para utilizar las variables de string de los TextBox, cuando empleemos ese contenido, habrá que hacerlo incluyendo .get() al final.

```
print mi_variable_string.get()
```

Uso de Checkbox

La síntaxis del widget checkbutton es la siguiente:

```
check = CheckButton(root, text="Lo que saldrá", var=variable
check.grid()
```

Donde check seria el nombre del widget, root el identificador, text el texto que aparece al

lado para informar que pasa o que seleccionas al seleccionar ese cuadrado y variable es la variable con la que se identifica el checkbutton.

Un ejemplo sencillo seria este:

```
var = IntVar()
check = Checkbutton(root, text="Hola mundo", variable=var)
check.grid()
```

Y ahora podemos saber si ese cuadrado ha sido seleccionado haciendo un if:

```
def funcion_presionada_por_button():
     if var.get():
          print "El checkbutton fue seleccionado"
     else:
          print "El checkbutton no fue seleccionado"
```

Otro ejemplo:

```
from Tkinter import *
def verificar():
        if var.get():
           print "El checkbutton fue seleccionado"
        else:
           print "El checkButton no fue seleccionado"
root = Tk()
var = IntVar()
check    =Checkbutton(root,    text="Seleccionado/    No    seleccionado",
variable=var)
cm =Button(root, text="Verificar", command=verificar, width=20)
check.grid()
cm.grid()
root.mainloop()
```

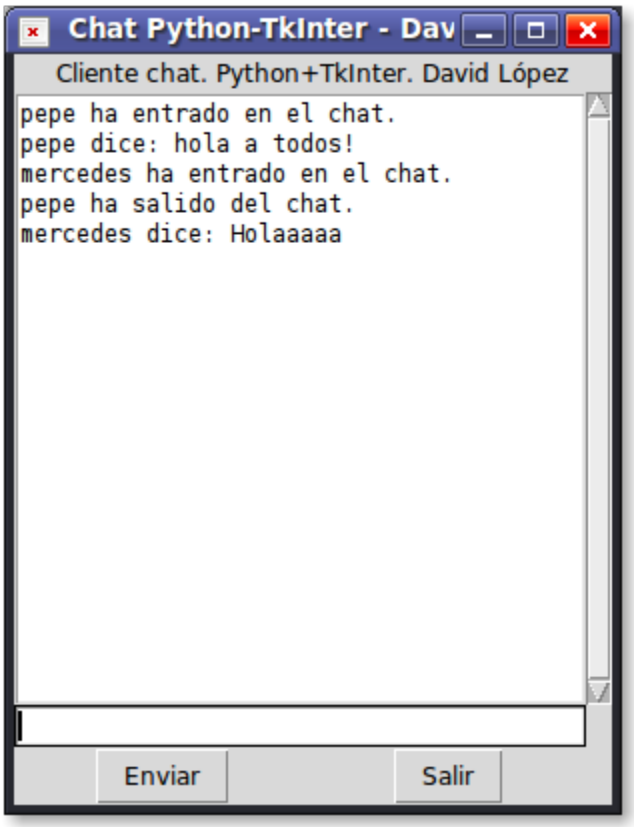
Cliente Chat en Python+TkInter

A continuación de presenta un ejemplo realizado en prácticas, un Chat con Sockets y Threads en Python. Se ha modificado el cliente para mostrarse con interfaz gráfica en TkInter.

Cliente

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
############################################################
#                                                          #
#Práctica7-Sockets-Chat. Python. Programación Avanzada     #
#             David López Fernández                        #
#           1º Ingeniería Informática                      #
#   Ingeniero Técnico en Informática de Sistemas           #
#             Universidad de Córdoba                       #
#                                                          #
############################################################
#Cliente

from Tkinter import *
import threading
import socket

class App:
       def __init__(self, master, socket):
             self.conn = socket
             self.frame = Frame(master)
             self.label=Label(self.frame,text="Clientechat. Python+TkInter. David López")

             self.textarea = Text(self.frame, height=20, width=40)
             self.scroll=Scrollbar(self.frame, command=self.textarea.yview)
             self.textarea.configure(yscrollcommand=self.scroll.set)

             self.texto_enviar = StringVar()
             self.text_ent=Entry(self.frame,textvariable=self.texto_enviar)

             self.btn_enviar=Button(self.frame,text="Enviar", command=self.enviar)
             self.btn_salir=Button(self.frame,text="Salir", command=self.salir)

             self.frame.grid()
             self.label.grid(row=0, column=0, columnspan=3)
             self.textarea.grid(row=1, column=0, columnspan=2)
             self.scroll.grid(row=1, column=2, sticky=N+S)
             self.text_ent.grid(row=2, column=0, columnspan=2, sticky=W+E)
             self.btn_enviar.grid(row=3, column=0)
             self.btn_salir.grid(row=3, column=1)

        def enviar(self):
             self.conn.send(self.text_ent.get())
             self.text_ent.delete(0, END)

        def salir(self):
             lee.parar()
             self.conn.send("END")
             self.frame.quit()

       def escribir(self, texto):
             self.textarea.insert(END, texto)

       class leer(threading.Thread):
          def __init__(self, socket):
             threading.Thread.__init__(self)
             self.mensaje = ''
```

```
            self.conn = socket
            self.stop = False
        def run(self):
            while (self.stop == False):
                  self.mensaje = self.conn.recv( 1024 )
                  app.escribir(self.mensaje+'\n')
            self.conn.close()
        def parar(self):
            self.stop = True


miSocket = socket.socket( socket.AF_INET, socket.SOCK_STREAM )
miSocket.connect( (socket.gethostname(), 9000 ) )

lee = leer(miSocket)

lee.start()
root = Tk()
root.title('Chat Python-TkInter - David López')
app = App(root, miSocket)

root.mainloop()
```



Servidor

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
############################################################
#                                                          #
#Práctica7-Sockets-Chat. Python. Programación Avanzada     #
#            David López Fernández                         #
#          1º Ingeniería Informática                       #
#   Ingeniero Técnico en Informática de Sistemas           #
#            Universidad de Córdoba                        #
#                                                          #
############################################################
#Servidor

import string
import threading
import socket

clientes = {'nombre':[], 'socket':[]}
  class gestionaClientes(threading.Thread):
       def __init__(self, socket):
       threading.Thread.__init__(self)
              self.conn = socket
              self.conectado = False
              self.data = ''
       def run(self):
          while True:
              self.data = self.conn.recv( 1024 )
              if 'ADD' in self.data:
                     #Busca si ya ha sido insertado
                     if(self.conectado == False):
                            self.conectado = True
                            clientes['nombre'].append(self.data[4:])
                            clientes['socket'].append(self.conn)
                            for i in clientes['socket']:
                                   if i != self.conn:
                            i.send(self.data[4:]+" ha entrado en el chat.")
                     else:
                            self.conn.send("Ya estás en el chat.")
                     print clientes
              if ('LIST' in self.data) and (self.conectado == True):
                     for i in clientes['nombre']:
                                   print "enviado a "+str(self.conn)+" "+str(i)
                                   self.conn.send(i+" ")
                     print clientes
              if ('END' in self.data):
                     if (self.conectado == True):
                            for i in clientes['socket']:
                                   if i == self.conn:
                                   nombre = clientes['nombre'][clientes['socket'].index(i)]

                                   clientes['nombre'].remove(nombre)
                                   clientes['socket'].remove(i)
                                   self.conectado = False

                            for i in clientes['socket']:
                                   if i != self.conn:
                                          i.send(nombre+" ha salido del chat.")
                                          self.conn.send(" ")
                                   else:
                                          self.conn.send(" ")
              if ('TEXT' in self.data and 'TEXT TO' not in self.data) and (self.conectado
== True):
                     for i in clientes['socket']:
                            if i != self.conn:
                            i.send(clientes['nombre']
```

```
[clientes['socket'].index(self.conn)]+" dice: "+self.data[5:])
                    print clientes
              if ('TEXT TO' in self.data) and (self.conectado == True):
       palabras = self.data[8:].split()
                    #busca si existe alguien con ese alias
                    for i in clientes['nombre']:
                           if i == palabras[0]:
                                  #Prepara el mensaje
                                  del palabras[0]
                                  mensaje = string.join(palabras, ' ')
                                  clientes['socket']
                                  [clientes['nombre'].index(i)].  send  (clientes['nombre']
[clientes['socket'].index(self.conn)]+" dice: "+mensaje)


       self.conn.close()

#creamos socket pasivo y escuchamos en el puerto 9000
s = socket.socket( socket.AF_INET, socket.SOCK_STREAM )
s.bind( ( socket.gethostname(), 9000 ) )
s.listen( 5 )
while(True):
       conn, addr = s.accept()
       gestionaClientes(conn).start()
```

2 PyGTK

Introducción

Posiblemente PyGTK sea la segunda opción más utilizada para la creación de interfaces gráficas con Python, solo por detrás de wxPython. PyGTK, es un binding de GTK, la biblioteca utilizada para desarrollar GNOME.

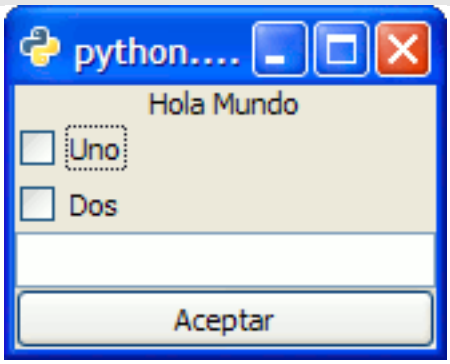
PyGTK cuenta con una API muy clara, limpia y elegante y es, además, muy sencillo de aprender, solo superado en ese aspecto por Tkinter. PyGTK también cuenta con grandes herramientas para construir la interfaz de forma gráfica, como Glade o Gazpacho.

Un punto negativo es que, hasta hace poco, era necesario instalar X11 para poder usar PyGTK en Mac OS, dado que GTK no había sido portado.

PyGTK se distribuye bajo licencia LGPL.

Este es el código de una aplicación de ejemplo en PyGTK:

```
import pygtk
import gtk

window = gtk.Window(gtk.WINDOW_TOPLEVEL)
window.connect("destroy",gtk.main_quit)

box = gtk.VBox(False, 0)
window.add(box)

label = gtk.Label("Hola Mundo")
c1 = gtk.CheckButton(label="Uno")
c2 = gtk.CheckButton(label="Dos")
entry = gtk.Entry()
button= gtk.Button("Aceptar")

box.add(label)
box.add(c1
box.add(c2)
box.add(entry)
box.add(button)

window.show_all()

gtk.main()
```



Al igual que TkInter, se puede realizar una interfaz gráfica para un programa escrito en Python escribiendo código, pero también se pueden realizar interfaces de manera más rápida, cómoda e intuitiva usando el diseñador de interfaces Glade.

A continuación se presenta un ejemplo utilizando cada una de estas opciones.

Un ejemplo escribiendo código

La idea es que al escribir un texto en la entrada de texto (el cuadro), y al presionar el botón OK (o la tecla ENTER) el texto se muestra en la etiqueta de la forma “Hola **texto_escrito**”.

El siguiente código se irá escribiendo en un archivo con extensión *.pl:

```
#! /usr/bin/env python
# -*- coding: UTF-8 -*-
```

Se importa el módulo pygtk y se le indica que use la versión 2 (en caso de que existan varias versiones de pygtk instaladas en el sistema)

```
import pygtk
pygtk.require("2.0")
```

Luego se importa el módulo de gtk para poder acceder a los #controles de Gtk+

```
import gtk
```

Se crea la clase que contiene la ventana principal del programa y los métodos de cada una las señales

```
class MainWin:
```

Primero se define como sera la ventana:

```
    def __init__(self):
```

Se crea una ventana toplevel (o sea que esta al frente de todas las ventanas) llamada "main_win" y se fija su titulo como "Ejemplo 1"

```
        main_win = gtk.Window(gtk.WINDOW_TOPLEVEL)
        main_win.set_title("Ejemplo 1")
```

A main_win se le conecta una señal (destroy), esto hará que cada vez que se presione el botón salir (la cruz del manejador de ventanas) se llamará al método on_quit que cerrara la ventana

```
        main_win.connect("destroy", self.on_quit)
```

Para agregar widgets (controles como botones, etiquetas, etc.) a la ventana, primero es necesario crear contenedores como cajas que contengan las widgets. En este ejemplo se crea una caja vertical con un espacio entre widgets de 5px y con la propiedad homogéneo en False

```
        vbox = gtk.VBox(False, 5)
```

Se crea una etiqueta con el texto "Hola mundo", se usa la palabra reservada "self" de python para poder hacer referencia a esta etiqueta desde otros métodos

```
        self.label = gtk.Label("Hola mundo")
```

Se crea un cuadro donde escribir (una entrada de texto en blanco) y luego le conectamos la señal "activate" que llama al método "on_button1_clicked", esto producirá que cuando se haga click en el botón Ok (que se creara mas adelante) la entrada de texto reaccione

```
        self.entry = gtk.Entry()
       self.entry.connect("activate",self.on_button1_clicked)
```

Ahora se crea el botón, que sera el botón OK del inventario de botones de GNOME.

Y luego se le indica al botón que cuando se haga click emita la señal "clicked" que llamará a "on_button1_clicked"

```
            button = gtk.Button(stock=gtk.STOCK_OK)
            button.connect("clicked", self.on_button1_clicked)
```

Ahora que ya creamos las widgets (la etiqueta, la entrada de texto y el botón) hay que añadirlos a la caja vertical creada anteriormente

Primero se le añade la etiqueta llamada label a la caja vertical

```
            vbox.add(self.label)
```

Luego se le añade al inicio de la segunda fila la entrada de texto activando las propiedades de expandir, rellenar y espaciado.

```
            vbox.pack_start(self.entry, True, True, 0)
```

Finalmente en la tercer fila se agrega el botón.

```
            vbox.pack_start(button, False, False, 0)
```

Ahora se agrega la caja vertical a la ventana y luego se muestra la caja (y todo lo que contiene) en la ventana principal.

```
            main_win.add(vbox)
            main_win.show_all()
```

Ahora dentro de la clase principal "MainWin" hay que definir qué hacen cada uno de los métodos que se llamaron anteriormente

Primero se define el método "on_button1_clicked"

```
    def on_button1_clicked(self, widget):
        #Primero obtiene el texto
        texto = self.entry.get_text()
       #Luego lo fija a la etiqueta en la forma"Hola
#texto".
        self.label.set_text("Hola %s" % texto)
```

Ahora se define el método "on_quit" que destruye la aplicación

```
    def on_quit(self, widget):
        gtk.main_quit()
```

Para terminar se inicia el programa

```
if __name__ == "__main__":
    #Iniciamos la clase.
    MainWin()
        #Además iniciamos el método gtk.main, que genera
     #un ciclo que se utiliza Para recibir todas las #señales
emitidas por los botones y demás
     #widgets.
    gtk.main()
```

Un ejemplo en usando Glade

Antes de empezar, hay que instalar pygtk y el soporte para glade, desde debian (y derivadas como ubuntu) es tan fácil como hacer un:

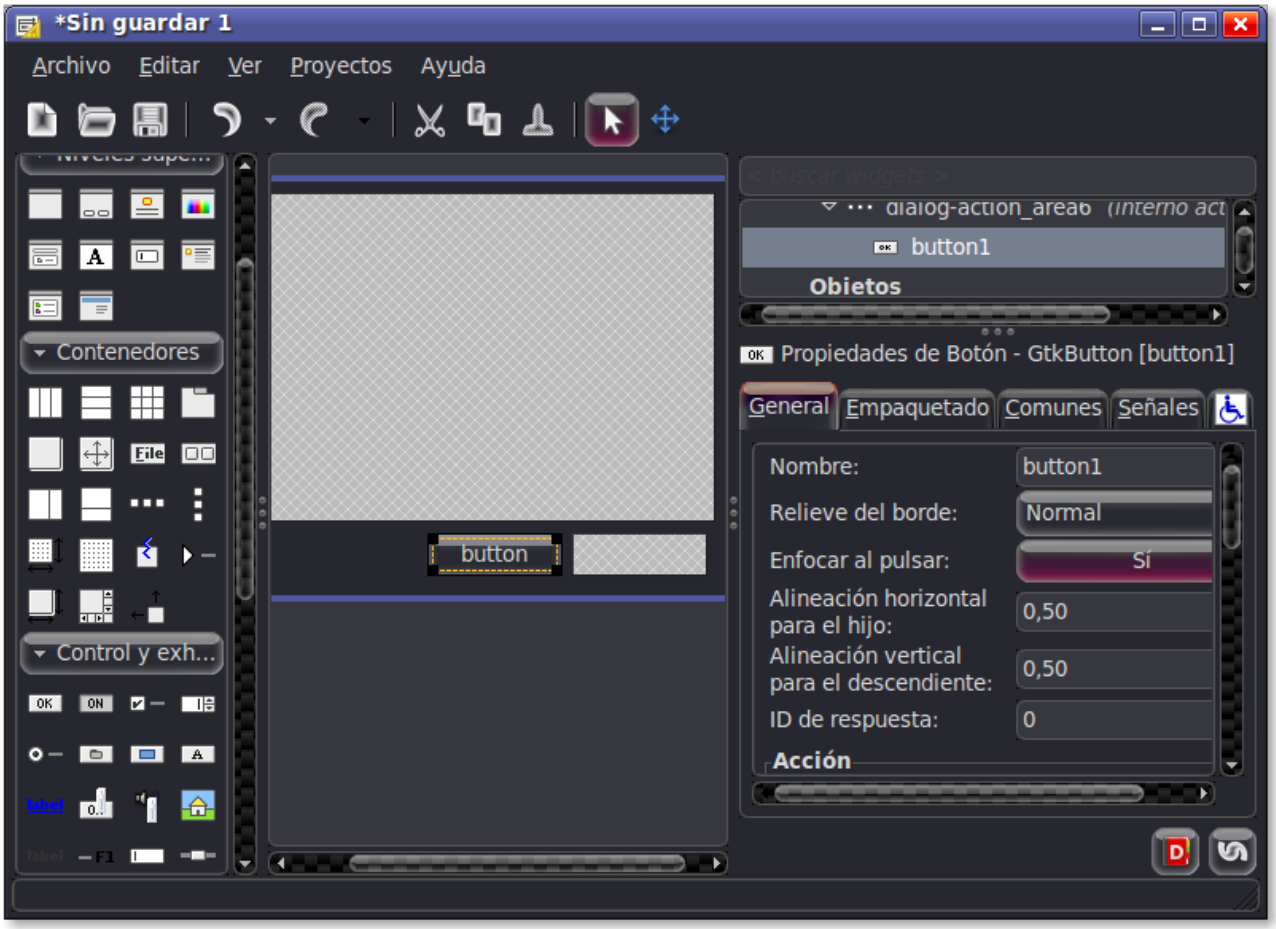
apt-get install python-gtk2 python-glade2

Además se necesita glade para crear la interfaz gráfica (Aunque no es necesario para ejecutarla), así que hay que escribir en consola:

apt-get install glade

En este caso se va a proceder a crear toda la interfaz gráfica usando glade, el cual crea un archivo con extensión *.glade en donde se guardan los nombres, señales, etc. de cada uno de los widget y de las ventanas. Así en el archivo .py solo se indica los widgets que se van a utilizar y que hacen los métodos.

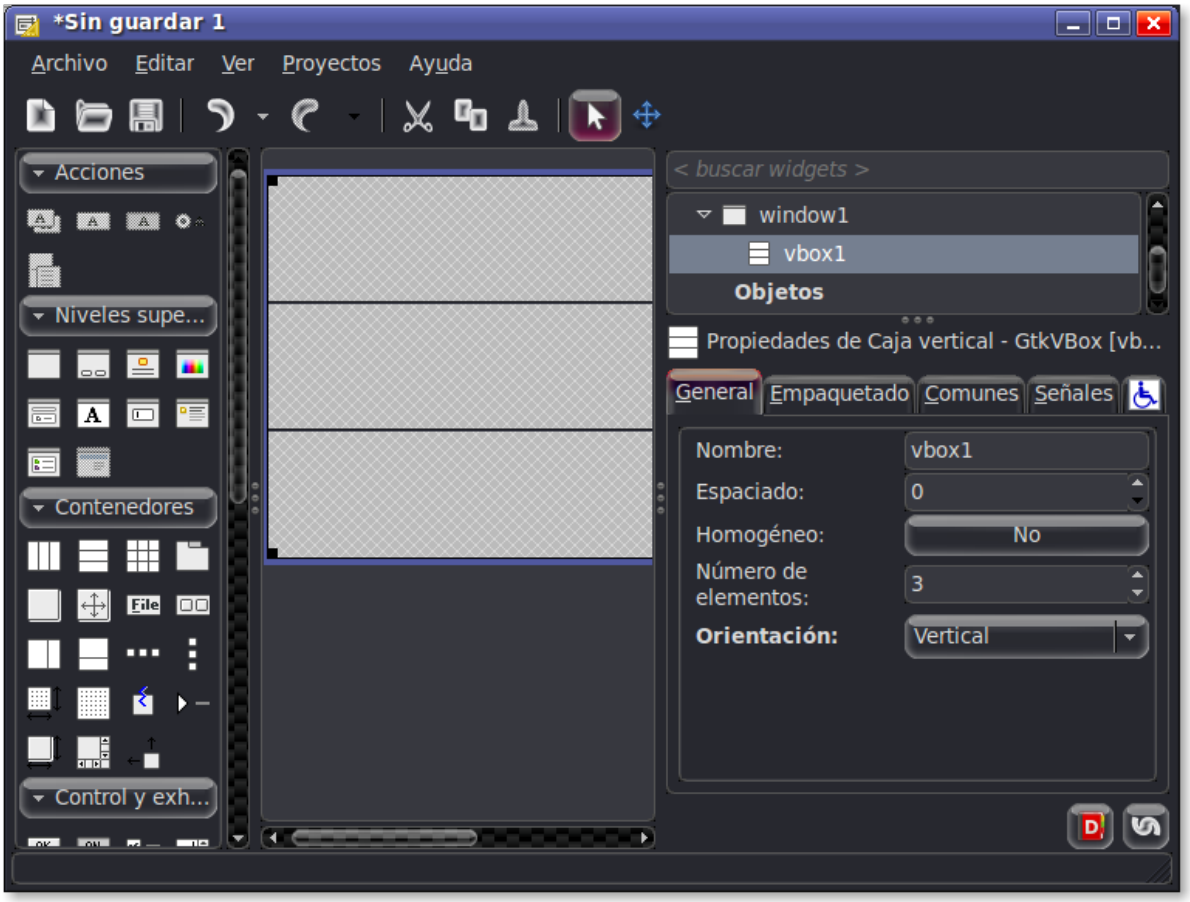
Primero hay que ejecutar glade y desde ventana principal crear un nuevo proyecto de GTK con Libglade. El programa tiene una paleta con los widgets que se pueden usar y una ventana de propiedades que muestra la información de cada widget y permite modificarla (ambas se ven la imagen a continuación), estas se usaran frecuentemente.



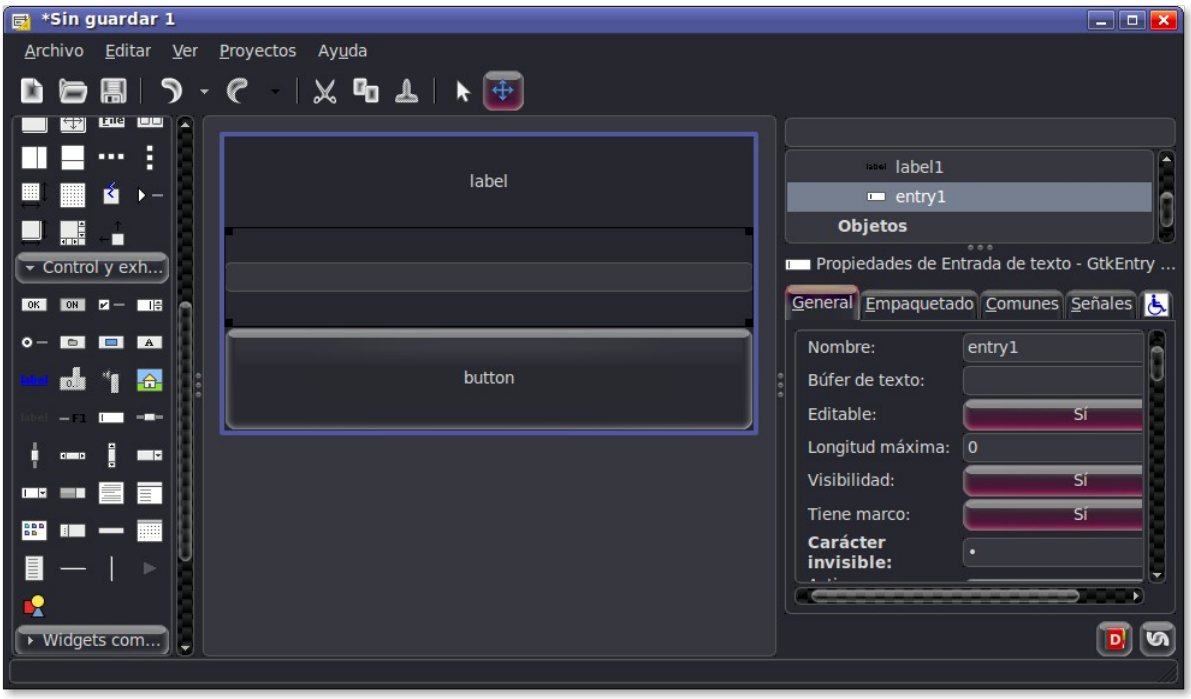
Ahora desde la paleta hacer click en el widget ventana (gtk.Window) para crear una ventana, luego seleccionar la ventana creada y modificar su información usando la ventana de

propiedades en donde se puede cambiar su nombre, titulo, ancho, largo, etc. en este caso el titulo será “Ejemplo 2″

El siguiente paso es crear la caja vertical, para eso en la paleta hacer click en el widget caja vertical (gtk.VBox), luego otro click en la ventana en donde se quiere colocarla y se deja con 3 filas. Queda esto:

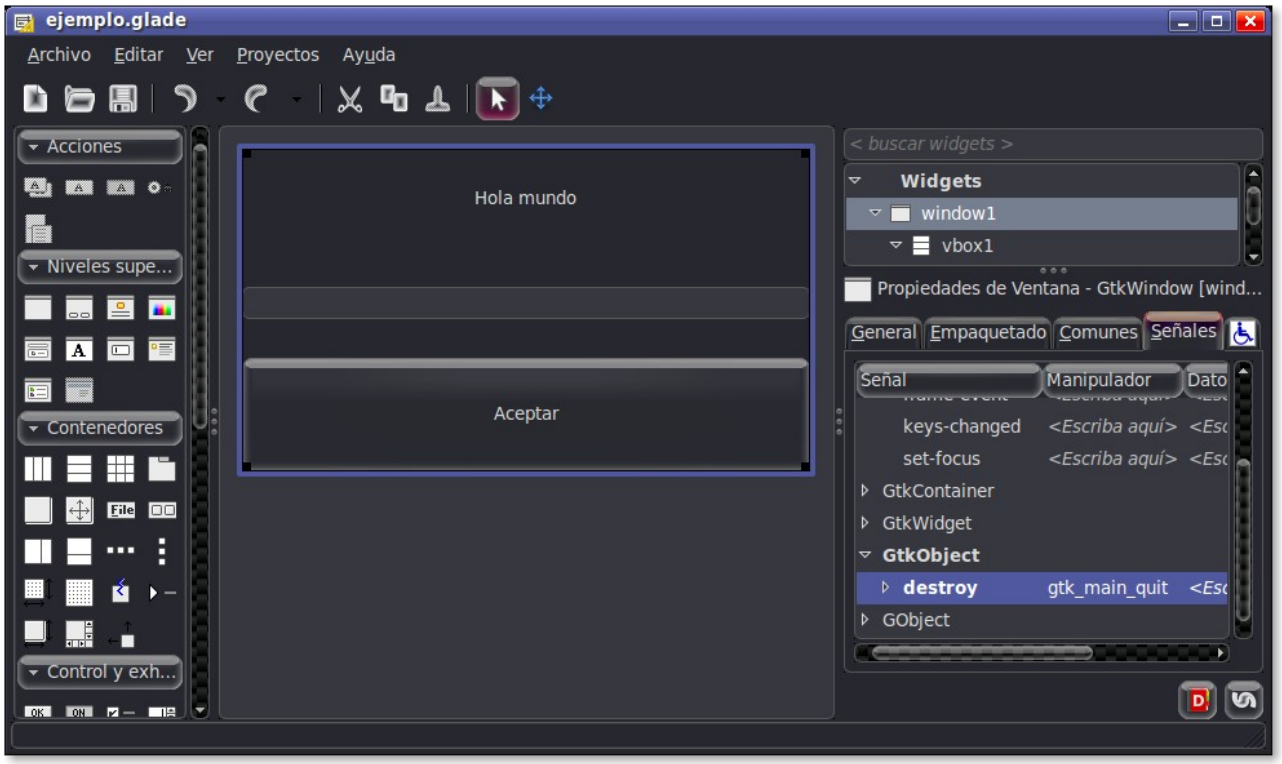


Ahora se inserta una etiqueta (gtk.Label) en la primer fila, después una entrada de texto (gtk.Entry) en la segunda fila y por último un botón (gtk.Button) en la tercera fila. Así queda una ventana como esta:



Ahora se cambian las propiedades de esta etiqueta para que el nombre sea “label1″ y el texto que muestra “Hola mundo”. En el caso de la entrada de texto (gtk.entry), su nombre será “entry1″ y el botón tendrá el nombre “button1″, y el texto “Aceptar”

El próximo paso es conectar las señales que emiten los eventos de cada widget, esto se hace en propiedades, en la pestaña “Señales”. Primero seleccionar la ventana (window1), buscar la señal “destroy” y seleccionar el manejador gtk_main_quit.



De igual manera seleccionar para el entry1 la señal “activate” y el manejador “on_button1_clicked”. Por último el para el button1 escoger la señal “clicked” y el manejador “on_button1_clicked”.

Finalmente hay que guardar el archivo con extensión .glade en el mismo directorio donde se vaya a crear el archivo de python.

Por ultimo el script queda así (los comentarios explican el código):

```
#! /usr/bin/env python
# -*- coding: latin1 -*-

# Importamos el módulo pygtk y le indicamos que use la versión 2
import pygtk
pygtk.require("2.0")

# Luego importamos el módulo de gtk y el gtk.glade, este ultimo que nos
# sirve para poder llamar/utilizar al archivo de glade
import gtk
import gtk.glade

# Creamos la clase de la ventana principal del programa
class MainWin:

    def __init__(self):
    #Le decimos a nuestro programa que archivo de glade usar (puede tener
    #un nombre distinto del script).Si no esta en el mismo directorio del
    #script habría que indicarle la ruta completa en donde se encuentra
      self.widgets = gtk.glade.XML("ejemplo.glade")

      #Creamos un pequeño diccionario que contiene las señales
      #definidas en glade y su respectivo método (o llamada)
```

```
      signals={"on_entry1_activate":self.on_button1_clicked,
"on_button1_clicked":self.on_button1_clicked,"gtk_main_quit" : gtk.main_quit }

        # Luego se auto-conectan las señales.
        self.widgets.signal_autoconnect(signals)
        #Nota: Otra forma de hacerlo es No crear el diccionario signals y
        #Solo usar "self.widgets.signal_autoconnect(self)"

          #Se muestran todos los widgets
          self.widgets.get_widget("window1").show_all()
        #Ahora obtenemos del archivo glade los widgets que vamos
        #a utilizar (en este caso son label1 y entry1)
        self.label1 = self.widgets.get_widget("label1")
        self.entry1 = self.widgets.get_widget("entry1")

     #Se definen los métodos, en este caso señales como           #"destroy" ya
      #fueron definidas en el .glade, así solo se necesita       #definir
"on_button1_clicked"
    def on_button1_clicked(self, widget):
        texto = self.entry1.get_text()
        self.label1.set_text("Hola %s" % texto)

# Para terminar iniciamos el programa
if __name__ == "__main__":
    MainWin()
    gtk.main()
```

Ahorcado en pyGTK+Glade

A continuación se explicará cómo realizar el diseño del interfaz gráfico de la aplicación "Ahorcado", realizada en prácticas. Para ello, se utilizará el diseñador de interfaces, Glade.

Antes de empezar, hay que instalar pygtk y el soporte para glade, como se explicó en el apartado anterior, haciendo:

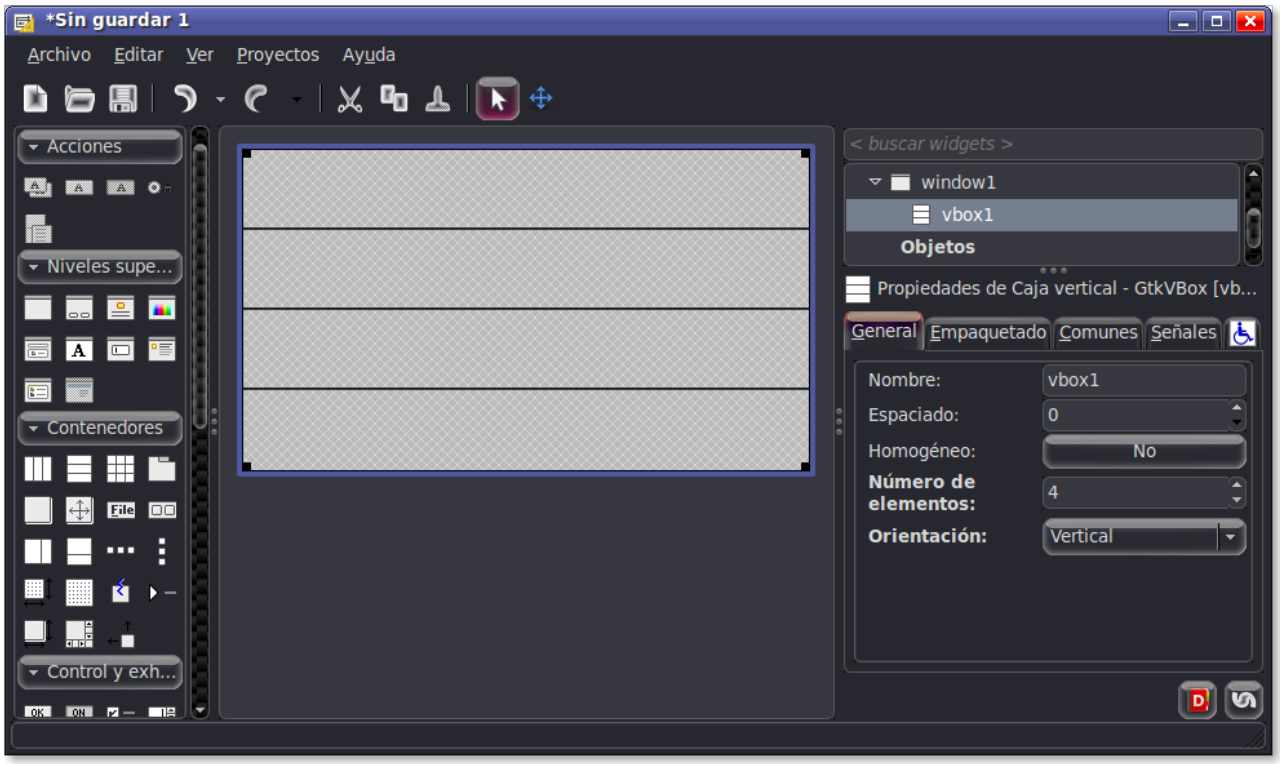
apt-get install python-gtk2 python-glade2

Además se necesita glade para crear la interfaz gráfica:

apt-get install glade

Primero hay que ejecutar glade y desde ventana principal crear un nuevo proyecto de GTK con Libglade.

Seleccionar el widget ventana, y ubicar en él una caja vertical de 4 elementos:

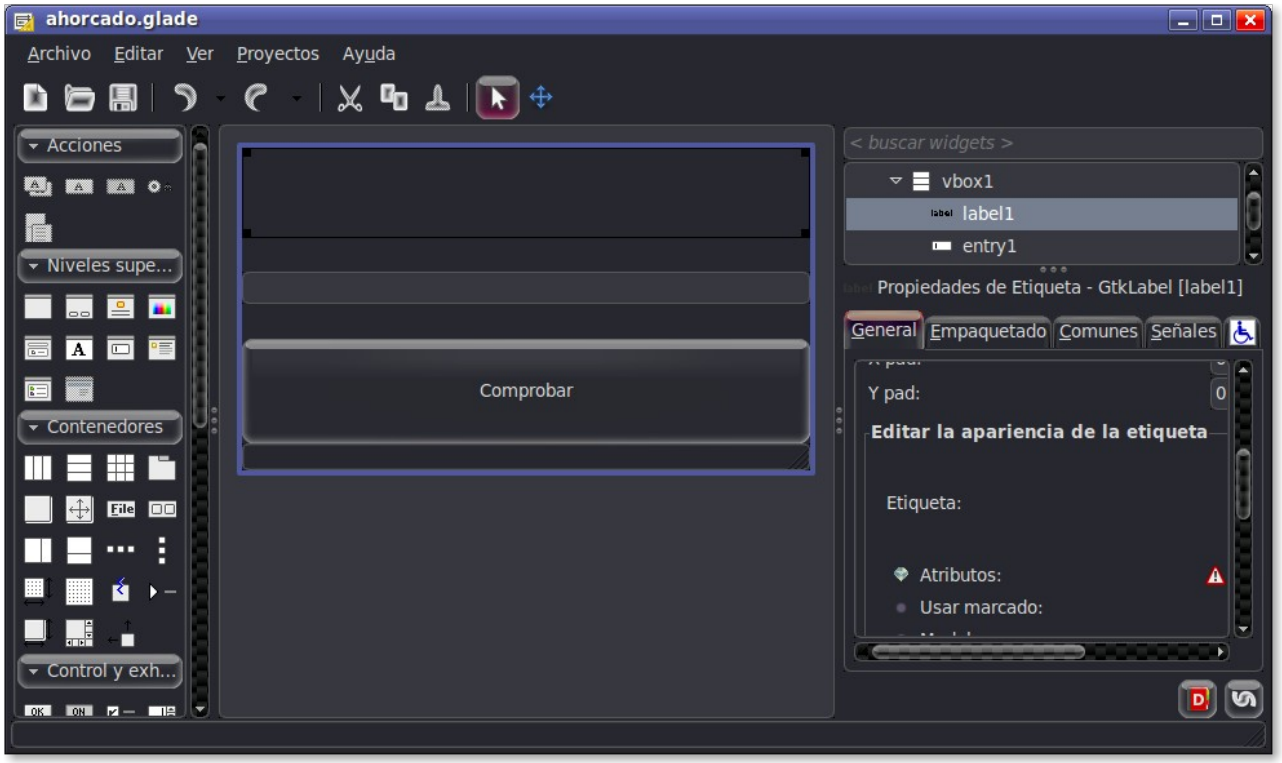


En el primero, poner un label. En el segundo una entrada de texto, en el tercero un botón, una barra de estados de un elemento en el cuarto.

Para el label, el nombre será "label1", estando la etiqueta por defecto inicializada a la cadena vacía. Para la entrada de texto, el nombre será "entry1", y para el botón el nombre será de "btn_comprobar", con el texto "Comprobar".

Modificar la ventana para mostrar una anchura predeterminada y una altura predeterminada de 200 y 50 píxeles correspondientemente.

Modificar también el widget “entry1” para limitar la longitud máxima a un carácter.



Ahora hay que conectar las señales, por lo que hay que seleccionar la ventana (“window1), y conectar la señal “destroy” con el manipulador “gtk_main_quit”, en la pestaña “Señales”.

Después hay que conectar la señal “clicked” del botón con el manipulador “on_btn_comprobar_clicked”, y la señal “activate” de la entrada de texto con la señal “on_btn_comprobar_clicked”, para que al presionar la tecla “enter” ocurra una pulsación del botón.

Guardar como ahorcado.glade con el formato de archivo libglade.

Por ultimo el script queda así (notar que se ha añadido una caja de diálogo para abrir el fichero de palabras):

```
#!/usr/bin/env python
# -*- coding: latin1 -*-

#############################################################
#                                                           #
#     Práctica 3. Python. Programación Avanzada             #
#             David López Fernández                         #
#           1º Ingeniería Informática                       #
#   Ingeniero Técnico en Informática de Sistemas            #
#             Universidad de Córdoba                        #
#                                                           #
#############################################################
import sys
import string
import random
import time
import pygtk
pygtk.require("2.0")
```

```
import gtk, gtk.glade

class App:

    def __init__(self):
      self.listaLetra = ''
      self.palabras = []
      self.nombre_archivo = ''
      self.pista = ''
      self.aciertos = 0
      self.fallos = 0


        self.glade = gtk.glade.XML("ahorcado.glade")

        self.glade.signal_autoconnect(self)
        self.glade.get_widget("window1").show_all()


      #accediendo a los controles

      self.btn_comprobar = self.glade.get_widget('btn_comprobar')
      self.label = self.glade.get_widget('label1')
      self.texto = self.glade.get_widget('entry1')
      self.statusbar1 = self.glade.get_widget('statusbar1')
      self.context_id = self.statusbar1.get_context_id('Barra estado')

      # Caja de diálogo para abrir un fichero
      # Check for new pygtk: this is new class in PyGtk 2.4
        if gtk.pygtk_version < (2,3,90):
          print "PyGtk 2.3.90 or later required for this example"
           raise SystemExit

        dialog = gtk.FileChooserDialog("Open..",
                                       None,
                                       gtk.FILE_CHOOSER_ACTION_OPEN,
                                       (gtk.STOCK_CANCEL, gtk.RESPONSE_CANCEL,
                                        gtk.STOCK_OPEN, gtk.RESPONSE_OK))
        dialog.set_default_response(gtk.RESPONSE_OK)

        filter = gtk.FileFilter()
        filter.set_name("All files")
        filter.add_pattern("*")
        dialog.add_filter(filter)

        filter = gtk.FileFilter()
        filter.set_name("Text")
        filter.add_mime_type("text/txt")
        filter.add_pattern("*.txt")
        dialog.add_filter(filter)

        response = dialog.run()
        if response == gtk.RESPONSE_OK:
             #Captura el nombre del fichero e inicializa el juego
             self.nombre_archivo = dialog.get_filename()
             self.palabras = devPalabras(self.nombre_archivo)
             self.palabra = seleccionaElemento(self.palabras)
             self.pista = '-'
             self.pista = self.pista*len(self.palabra)
             self.label.set_label(str(self.pista))
        elif response == gtk.RESPONSE_CANCEL:
           #Sale del diálogo y de la aplicación
             gtk_main_quit()
        dialog.destroy()


    def on_window1_delete_event(self, widget, event):
        gtk.main_quit()
```

```
    def on_btn_comprobar_clicked(self, widget):
      letra = self.texto.get_text()
      if(letra == ''):
            return
      listaLetra=buscaLetra(letra, self.palabra)
      if (listaLetra) :
            self.aciertos += 1
      else :
            self.fallos += 1
      for i in listaLetra:
                  self.pista=self.pista[0:i]+letra+self.pista[i+1:]
        self.label.set_label(self.pista)
      if(self.pista != self.palabra):
            message_id=self.statusbar1.push(self.context_id,         str(self.aciertos)+'
aciertos, '+str(self.fallos)+' fallos.')
      else:
            message_id=self.statusbar1.push(self.context_id,'FIN.  '+str(self.aciertos)+'
aciertos, '+str(self.fallos)+' fallos.')
      self.texto.set_text("")



def devPalabras( nombreFichero ):
      f = open(nombreFichero, "r")
      contenido = f.read()
      palabras = contenido.split()
      f.close
      #Elimino el contenido repetido
      f = open(nombreFichero, "w")
      nuevaLista = []
      for i in palabras:
            if nuevaLista.count(i) == 0:
                  nuevaLista.append(i)
      contenido = string.join(nuevaLista, '\n')
      f.write(contenido)
      f.close
      return palabras

def seleccionaElemento(palabras):
      return random.choice(palabras)

def buscaLetra(letra, cadena):
      posicion = 0
      lista = []
      while cadena.find(letra, posicion, len(cadena)) != -1 :
            posicion = string.find(cadena, letra, posicion, len(cadena))
            lista.append(posicion)
            posicion+=1
      return lista

if __name__ == "__main__":
      try:
            a = App()
            gtk.main()
      except KeyboardInterrupt:
            pass
```

3 PyQT

Introducción

Es el menos popular, aunque es un toolkit sencillo de utilizar y con muchas posibilidades. Es especialmente interesante para el desarrollo en KDE, dado que Qt es la librería utilizada para crear este entorno.

No obstante el interés en Qt no se limita a KDE, sino que es una biblioteca multiplataforma que, además, desde la versión 4, utiliza widgets nativos para las distintas plataformas (anteriormente Qt emulaba el aspecto de la plataforma en la que corría).

Como aplicación de RAD se puede utilizar Qt Designer.

PyQt utiliza un modelo de licencias similar al de Qt, con una licencia dual GPL/PyQt Commercial. Si el programa a desarrollar es compatible con la licencia GPL, se puede usar PyQt sin más preocupaciones. En caso contrario hay que pagar para obtener una licencia comercial.

El código de una aplicación de ejemplo en PyQt tendría el siguiente aspecto:

```
import sys
from PyQt4.QtCore import *
from PyQt4.QtGui import *

class Form(QWidget):
    def __init__(self):
        QWidget.__init__(self)

        layout = QVBoxLayout()
        layout.addWidget(QLabel("Hola mundo"))
        layout.addWidget(QCheckBox("Uno"))
        layout.addWidget(QCheckBox("Dos"))
        layout.addWidget(QLineEdit())
        layout.addWidget(QPushButton("Aceptar"))

        self.setLayout(layout)

app = QApplication(sys.argv)
form = Form()
form.show()
app.exec_()
```

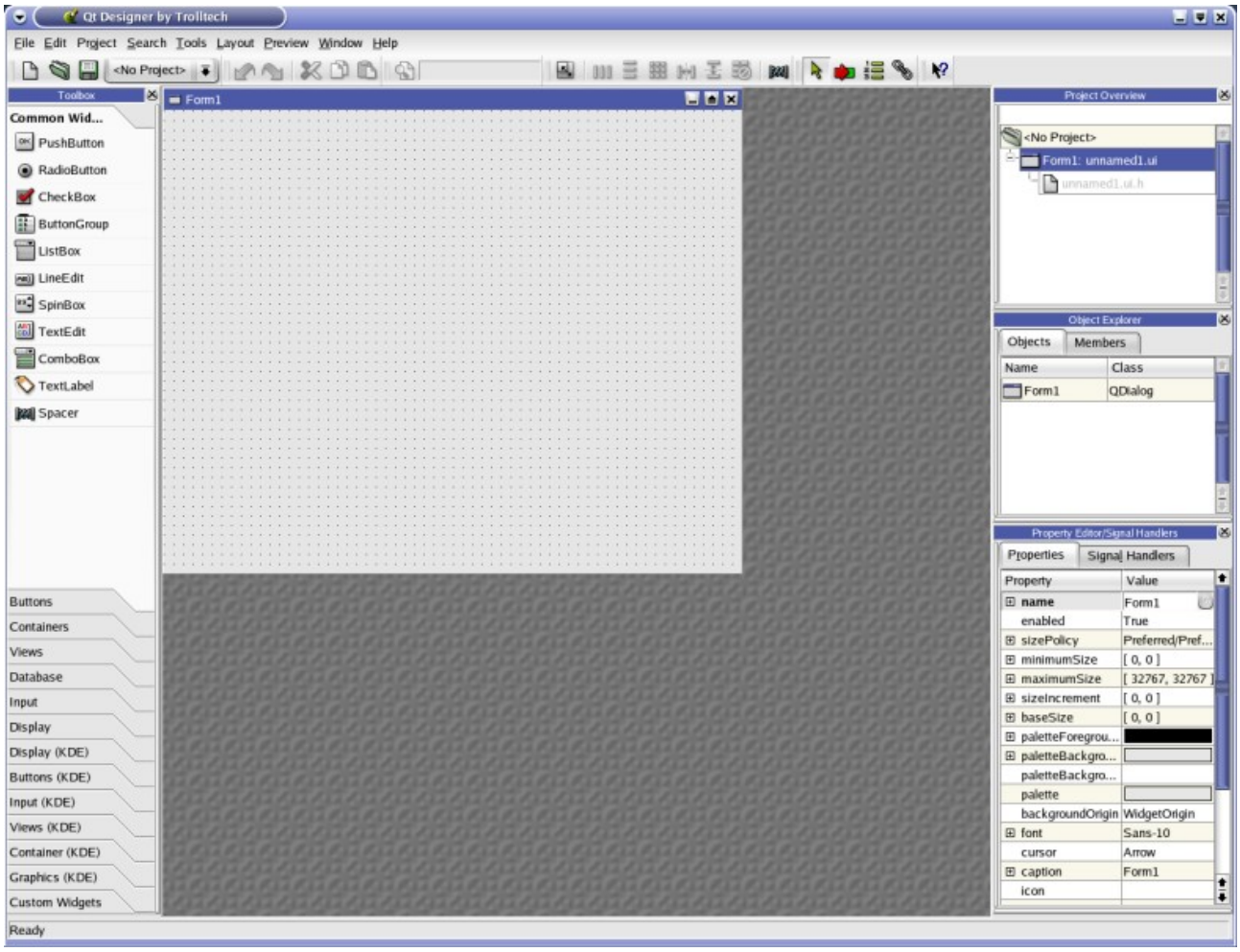
pyQt+QtDesigner

Al igual que ocurre con PyGTK y Glade, también se puede usar pyQT con un diseñador de interfaces, en este caso, con QtDesigner. Para ello, se necesita:

- Python-qt-dev
- Python-qt3
- pyqt-tools

Se va a proceder a diseñar una sencilla aplicación “Hola Mundo”.

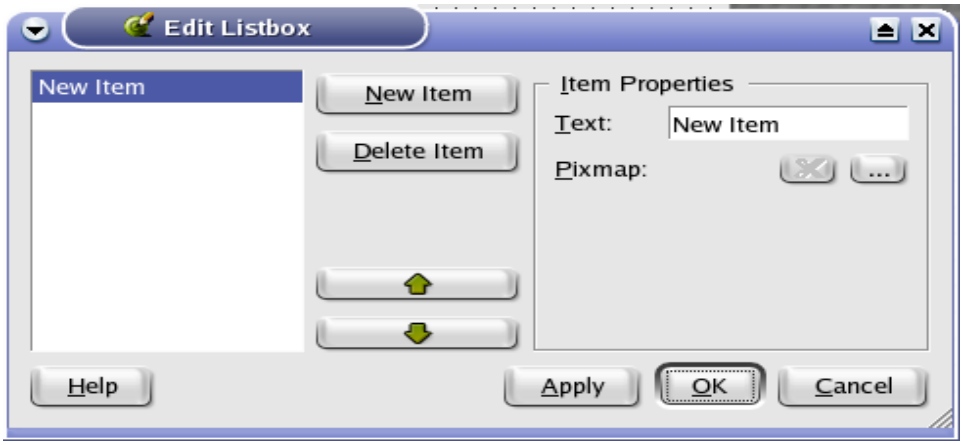
Comenzar abriendo QtDesigner. Aparecerá una ventana como la que sigue:



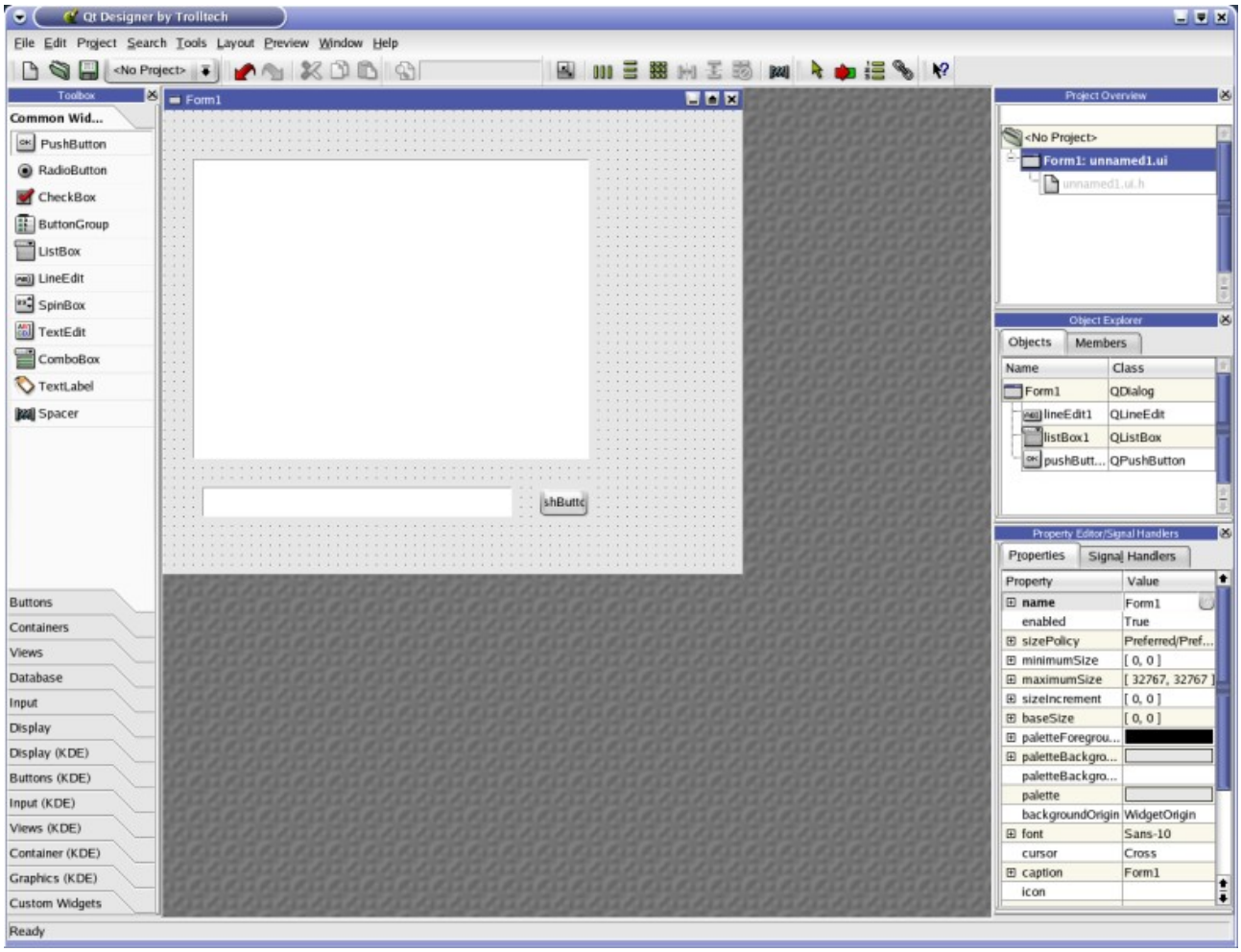
Seleccionar New File/Project => dialog, y presionar Ok. Aparecerá una ventana como esta.

De la barra de herramientas de la izquierda, seleccionar LineEdit, e insertarlo en el formulario con el nombre LineEdit1.

Después insertar una lista (ListBox) con el nombre listBox1. Hay que borrar el elemento que contiene por defecto, por lo que se edita, presionando sobre Delete Item:



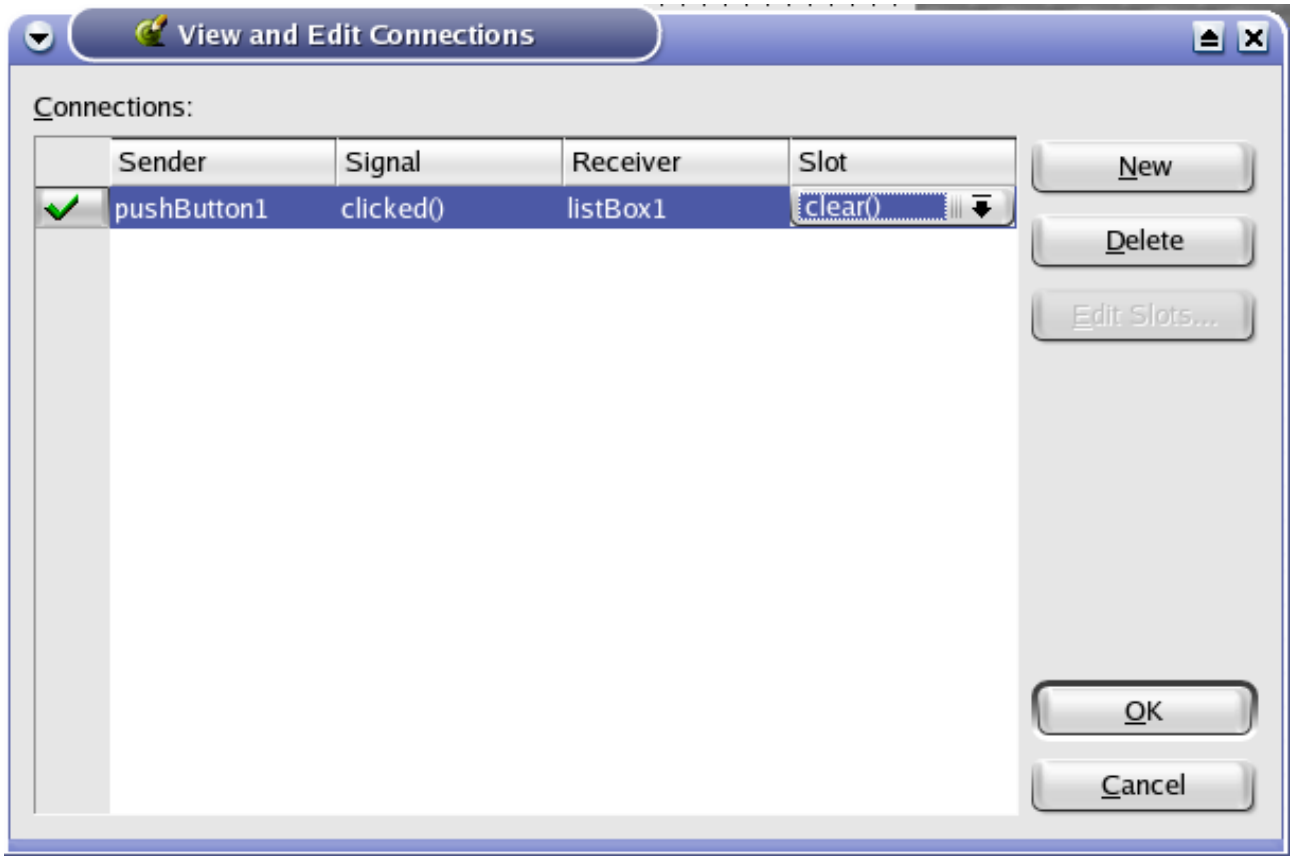
Ahora hay que insertar un PushButtom, seleccionándolo de la caja de herramientas y llevándolo hasta el formulario. Nombrarlo como pushButton1, con la caption “X”.



Ahora hay que conectar los signals (eventos que se disparan cuando se interacciona con un control) con los slots, o funciones correspondientes. Para ello, hay que hacer click en el botón con la flecha roja y el rectángulo verde de la barra superior.

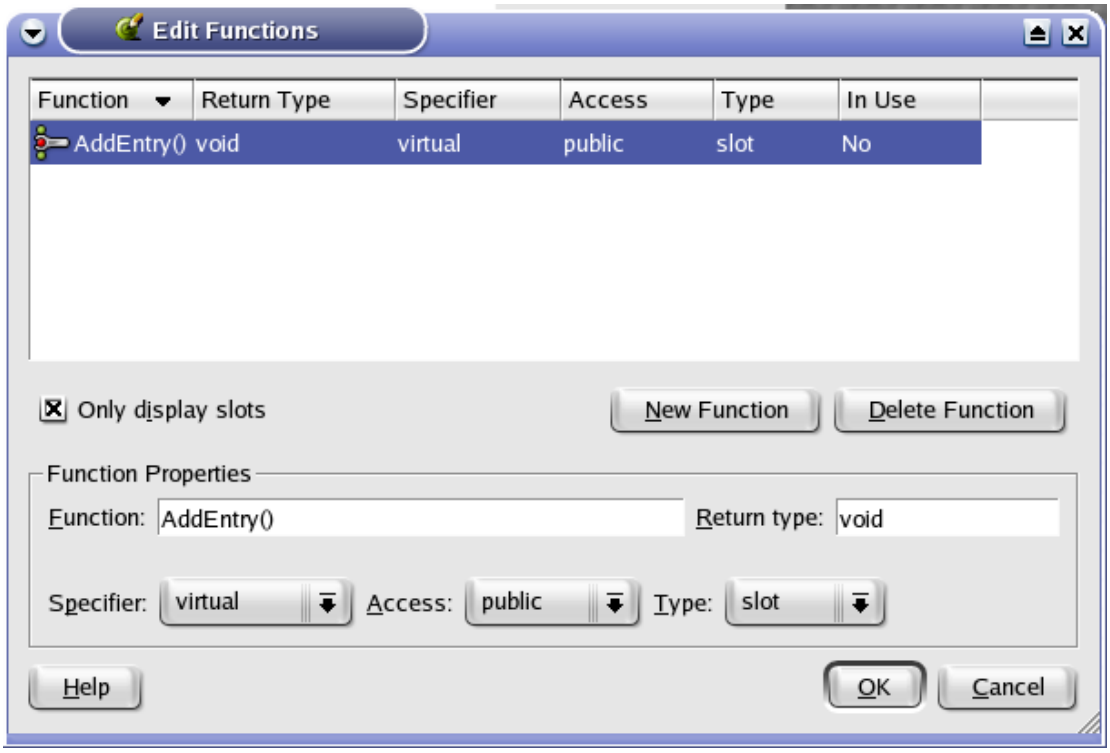
Tras esto, aparecerá una ventana como la siguiente, que en la que hay que introducir los

datos que aparecen en la imagen. Esto es, introducir en el campo “Sender” el “pushButton1”, en “Signal” la señal “clicked()”, en “Receiver” el “listBox1”, y en slot “Clear()”. (Esto borrará todo el contenido de la lista tras hacer click en el botón X).

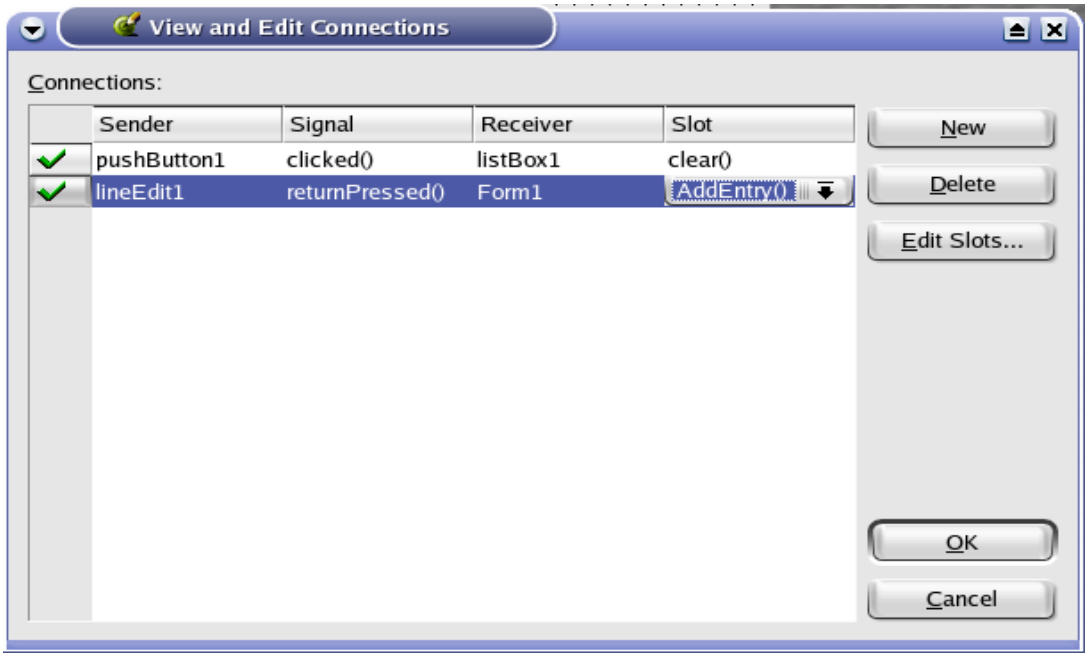


Ahora se va a proceder a crear un slot propio, sin utilizar los built-in. Este slots basicamente introducirá el texto que se haya escrito en la caja de texto tras presionar la tecla “Enter”. Para esto, Abrir “Edit/Slots”.

Un dialogo se mostrará ahora. Inicialmente la lista estará vacía. Se añade un nuevo slot haciendo click en “New Function”. Entonces llamar a la función AddEntry(), sin cambiar nada más, y hacer click Ok. (Posteriormente se escribirá el código).



Ahora que tenemos el slot, podemos conectar algún control a él. Volvemos a editar los signals, en la ventana de "View and Edit conections", y se introduce como "Sender" el "lineEdit1", a la señal "returnPressed()", en "receiver" introducir "Form1", siendo el slot es el que se acaba de crear, AddEntry().



Finalmente, se escribe el código. Para ello hay que abrir la ventana "Project Overview",

situada en la esquina superior derecha, y hacer doble click en “form1.ui.h”.

Introducir el siguiente código:

```
void Form1::AddEntry()
{
     #get the text typed in the line edit
     e=self.lineEdit1.text().ascii()
     #add that text to the list box
     self.listBox.insertItem(e)
     #clear the line edit
     self.line.Edit1.clear()
}
```

Ya está casi todo hecho, sólo falta poner lo propiedad autoDefault del botón en “False”, para evitar que se pulse al pulsar la tecla Enter. Esto puede hacerse desde la ventana de propiedades del propio botón. Guardar el fichero como “form1” en el directorio del fichero python.

Tras esto, hay que compilar el código del GUI en código Python. Introducir en una consola:

```
pyuic form1.ui > form1.py
```

El código del script en Python es el siguiente:

```
from qt import *
from form1 import *
import sys
if __name__ == "__main__":
     app = Qapplication(sys.argv)

     f = Form1()

     f.show()
     app.setMainWidget(f)
     app.exec_loop()
```

4 WxPython

Introducción

WxPython es un wrapper open source para el toolkit anteriormente conocido como wxWindows: wxWidgets. wxPython es posiblemente el toolkit para desarrollo de interfaces gráficas en Python más popular, superando incluso a Tkinter.

WxPython cuenta con más y mejores widgets que TKinter, y ofrece un muy buen aspecto en todas las plataformas, utilizando MFC en Windows y GTK en Linux.

Cuenta además con herramientas muy interesantes como wxGlade, una aplicación RAD para diseñar las interfaces gráficas de forma visual.

Sin embargo, la API adolece de una cierta falta de consistencia y un estilo muy alejado de Python y más cercano a C++, ya que, de hecho, uno de sus objetivos es no distanciarse demasiado del estilo de wxWidgets.

Tanto wxPython como wxWidgets se distribuyen bajo una licencia “wxWindows Licence”, que consiste esencialmente en una LGPL con la excepción de que las obras derivadas en formato binario se pueden distribuir como el usuario crea conveniente.

Una aplicación que use wxPython tendría el siguiente aspecto:

```
import wx

class Frame(wx.Frame):
    def __init__(self):
        wx.Frame.__init__(self, parent=None)
        panel = wx.Panel(self)
        text = wx.StaticText(panel, -1, "Hola mundo")
        c1 = wx.CheckBox(panel, -1, label="Uno")
        c2 = wx.CheckBox(panel, -1, label="Dos")
        t = wx.TextCtrl(panel)
        b1 = wx.Button(panel, -1, label="Aceptar")

        sizer = wx.BoxSizer(wx.VERTICAL)
        sizer.Add(text, 0, wx.ALL, 0)
        sizer.Add(c1, 0, wx.ALL, 0)
        sizer.Add(c2, 0, wx.ALL, 0)
        sizer.Add(t, 0, wx.ALL, 0)
        sizer.Add(b1, 0, wx.ALL, 0)
        panel.SetSizer(sizer)
        panel.Layout()

app = wx.App(redirect=True)
Frame().Show()
app.MainLoop()
```

Primeros pasos a seguir

- Importación del modulo

```
#!/usr/bin/python
try:
    import wx
except ImportError:
    raise ImportError,"Se requiere el modulo wxPython
```

- Creación de una clase

```
#!/usr/bin/python
try:
    import wx
except ImportError:
    raise ImportError,"Se requiere el modulo wxPython"
class simpleapp_wx(wx.Frame):
```

- El constructor

Un GUI es una jerarquía de objetos, un botón puede estar contenido en un panel que figura en una solapa que es contenido en una ventana, etc.

Por lo tanto, cada elemento gráfico tiene un padre (su contenedor, por lo general). Por lo tanto también tiene un constructor padre (parent) como parámetro. Hacer un seguimiento de los padres es muy útil cuando hay que mostrar/ocultar un grupos de widgets, redibujar la pantalla o simplemente destruirlas cuando la aplicación termina. Asi, simpleapp_wx hereda de wx.Frame, por lo que tenemos que llamar al constructor wx.Frame (wx.Frame.__init__ ()).

```
#!/usr/bin/python
try:
    import wx
except ImportError:
     raise ImportError,"Se requiere el modulo wxPython"
class simpleapp_wx(wx.Frame):
    def __init__(self,parent,id,title):
        wx.Frame.__init__(self,parent,id,title)
```

El objeto wx.Frame tiene dos parámetros: id (identificador del widget) y title (el título de la ventana).

- Mantener relacion con el padre

Es una buena costumbre, cuando se realiza la construcción de cualquier tipo de componente GUI, mantener una referencia a nuestra clase padre.

```
#!/usr/bin/python
try:
    import wx
except ImportError:
    raise ImportError,"Se requiere el modulo wxPython"
class simpleapp_wx(wx.Frame):
    def __init__(self,parent,id,title):
        wx.Frame.__init__(self,parent,id,title)
        self.parent = parent
```

- Inicializacion del GUI

Usualmente es mejor tener la porción de código que crea todos los elementos GUI (botón, campos de texto …) separado de la lógica del programa. Es por eso que se crea el metodo initialize(). Se va a crear todos los widgets (botones, campo de texto, etc) en este método.

En wxPython tenemos que usar self.Show(True) para forzar que la ventana se muestre en pantalla (de lo contrario permanecerá oculta).

```
#!/usr/bin/python
try:
    import wx
except ImportError:
    raise ImportError,"Se requiere el modulo wxPython"
class simpleapp_wx(wx.Frame):
    def __init__(self,parent,id,title):
        wx.Frame.__init__(self,parent,id,title)
        self.parent = parent
        self.initialize()
    def initialize(self):
        self.Show(True)
```

- Creacion de la clase principal

Ya está la clase completamente funcional y sintacticamente correcta.

Se crea una función main que se ejecuta cuando el programa se invoca desde la línea de comandos.

En wxPyhon, es obligatoria la instanciación de un objeto wx.App () antes de la creación de los elementos del GUI. Por lo que se hace app = wx.App (). Luego se instancia la clase (frame = simpleapp_wx ()). Asimismo, no le asigna ningún padre (None), porque es el primer elemento GUI que vamos a construir. Se usa -1 a dejar a wxPython elegir un identificador (id) propio. Y da a la ventana un título: «Mi aplicación».

```
#!/usr/bin/python
try:
    import wx
except ImportError:
    raise ImportError,"Se requiere el modulo wxPython"
class simpleapp_wx(wx.Frame):
    def __init__(self,parent,id,title):
        wx.Frame.__init__(self,parent,id,title)
        self.parent = parent
        self.initialize()
    def initialize(self):
        self.Show(True)

if __name__ == "__main__":
    app = wx.App()
    frame = simpleapp_wx(None,-1,'Mi aplicación')
```

- Entrando en el loop

Ahora, hay que decir al programa que entre en un bucle esperando eventos, esto se hace con mainloop ()

Esto significa que cada programa entra en un bucle "infinito", a la espera de eventos (ej. clic en un botón, pulsacion de una tecla, alguna acción con el ratón, etc.) En wxPython el bucle principal recibirá estos eventos y los manejara en consecuencia. Esto se llama programación event-driven o controlada por eventos (debido a que el programa no hará nada, sino esperar los eventos, y sólo reaccionara cuando se reciba uno de ellos).

```
#!/usr/bin/python
try:
    import wx
except ImportError:
    raise ImportError,"Se requiere el modulo wxPython"
class simpleapp_wx(wx.Frame):
    def __init__(self,parent,id,title):
        wx.Frame.__init__(self,parent,id,title)
        self.parent = parent
        self.initialize()
    def initialize(self):
        self.Show(True)

if __name__ == "__main__":
    app = wx.App()
    frame = simpleapp_wx(None,-1,'Mi aplicación')
    app.MainLoop()
```

Ya está listo el programa para ser ejecutado.

- Diseño de interfaz

Hay varias manera de incrustar widgets en una ventana (o en otro contenedor): Añadir apilados horizontalmente, verticalmente, en una rejilla, etc

Existen diferentes clases, llamadas gestores de diseño (layout managers), capaces de colocar los widgets en los contenedores de diferentes maneras. Algunos son más flexibles que otros. Cada contenedor (ventana, panel, solapa, diálogo …) puede tener su propio gestor de diseño.

En wxPython, para crear el gestor de diseño del tipo rejilla se usa grilla=wx.GridBagSizer() y luego se indica a la ventana que lo use (self.SetSizerAndFit(grilla)).

```
#!/usr/bin/python
try:
    import wx
except ImportError:
    raise ImportError,"Se requiere el modulo wxPython"
class simpleapp_wx(wx.Frame):
    def __init__(self,parent,id,title):
        wx.Frame.__init__(self,parent,id,title)
        self.parent = parent
        self.initialize()
    def initialize(self):
        grilla = wx.GridBagSizer()
        self.SetSizerAndFit(grilla)
        self.Show(True)
```

```
if __name__ == "__main__":
    app = wx.App()
    frame = simpleapp_wx(None,-1,'Mi aplicacion')
    app.MainLoop()
```

Se obtiene una ventana mínima que puede ser redimensionada.

Agregar un widget de entrada de texto

Los pasos para agregar un widget a un contenedor son dos, primero definirlo para crearlo, luego agregarlo a un gestor de diseño.

Crear del widget

El objeto que va a ser creado es una entrada de texto en wxPython esa clase se llama TextCtrl (self.entrada=wx.TextCtrl()), se pasa como parámetro self y el padre, porque la ventana será el padre de este widget, es decir que aparecerá dentro de esta ventana.

TextCtrl recibe dos parámetros además de self, -1 (para que wxPython asigne automáticamente un identificador), y el texto a mostrar (value=u"Enter text here.").Agregar el widget

Agregar el widget

Ahora, lo agregaremos a nuestros gestor de diseño (GridBagSizer).

Llamamos al método .Add() de nuestra grilla, le pasamos el widget que acaba de crear (self.entrada) y sus coordenadas de ubicación en la grilla (0,0) y (1,1) que es el espaciado alrededor del widget. En nuestro caso, el widget no se extienden a varias celdas, wx.EXPAND le dice al gestor de diseño (grilla) que debe ampliar la entrada de texto si su celda es redimensionada.

```
#!/usr/bin/python
try:
    import wx
except ImportError:
    raise ImportError,"Se requiere el modulo wxPython"
class simpleapp_wx(wx.Frame):
    def __init__(self,parent,id,title):
        wx.Frame.__init__(self,parent,id,title)
        self.parent = parent
        self.initialize()
    def initialize(self):
        grilla = wx.GridBagSizer()
    self.entrada = wx.TextCtrl(self,-1,value=u"Ingrese un texto:")
        grilla.Add(self.entrada,(0,0),(1,1),wx.EXPAND)
        self.SetSizerAndFit(grilla)
        self.Show(True)

if __name__ == "__main__":
    app = wx.App()
    frame = simpleapp_wx(None,-1,'Mi aplicación')
    app.MainLoop()
```

Agregar un botón

```
#!/usr/bin/python
try:
    import wx
except ImportError:
    raise ImportError,"Se requiere el módulo wxPython"
class simpleapp_wx(wx.Frame):
    def __init__(self,parent,id,title):
        wx.Frame.__init__(self,parent,id,title)
        self.parent = parent
        self.initialize()
    def initialize(self):
        grilla = wx.GridBagSizer()
     self.entrada = wx.TextCtrl(self,-1,value=u"Ingrese un texto:")
        grilla.Add(self.entrada,(0,0),(1,1),wx.EXPAND)
        boton = wx.Button(self,-1,label="Pulse aquí")
        grilla.Add(boton, (0,1))
        self.SetSizerAndFit(grilla)
        self.Show(True)

if __name__ == "__main__":
    app = wx.App()
    frame = simpleapp_wx(None,-1,'Mi aplicación')
    app.MainLoop()
```

Agregar una etiqueta

Se trata de un StaticText en wxPython. Para poner colores, hay que llamar dos métodos para ello (SetForegroundColour y SetBackgroundColour), el texto estará alineado a la izquierda por defecto, se ubica la etiqueta en el gestor de diseño, la rejilla, pero esta vez el nuevo widget se extiende a través de dos celdas (de modo que aparezca por debajo del campo de texto y del botón.), para ello se debe especificar un área de (1,2), es decir una celda verticalmente, y dos celdas horizontalmente. Finalmente para la expansión de la etiqueta usamos wx.EXPAND.

```
#!/usr/bin/python
try:
    import wx
except ImportError:
    raise ImportError,"Se requiere el módulo wxPython"

class simpleapp_wx(wx.Frame):

   def __init__(self,parent,id,title):
        wx.Frame.__init__(self,parent,id,title)
        self.parent = parent
        self.initialize()

   def initialize(self):
        grilla = wx.GridBagSizer()
    self.entrada = wx.TextCtrl(self,-1,value=u"Ingrese un texto:")
        grilla.Add(self.entrada,(0,0),(1,1),wx.EXPAND)
        boton = wx.Button(self,-1,label="Pulsa aquí")
        grilla.Add(boton, (0,1))
        self.etiqueta = wx.StaticText(self,-1,label=u'Hola !')
        self.etiqueta.SetBackgroundColour(wx.BLUE)
        self.etiqueta.SetForegroundColour(wx.WHITE)
        grilla.Add( self.etiqueta, (1,0),(1,2), wx.EXPAND )
        self.SetSizerAndFit(grilla)
        self.Show(True)

if __name__ == "__main__":
    app = wx.App()
    frame = simpleapp_wx(None,-1,'Mi aplicación')
    app.MainLoop()
```

Manejador de eventos y modificación de atributos

Los manejadores de eventos (event handler) son métodos que se llaman cuando algo sucede en el GUI. Se relacionan en el manejador de eventos widgets específicos con eventos puntuales. De esta forma se especifica que es lo que tiene que hacer la aplicación ante cualquier evento que pueda ser capturado, por ejemplo, cuando se hace clic con ratón sobre el botón o se presiona ENTER en el campo de texto.

Se crea un método SiCliqueaBoton(), método que se llama cuando se hace clic en el botón. También se crea un método SiPulsamosEnter(), método que se llamará luego de presionar ENTER en el campo de texto. Observar que es necesario agregar en la entrada el parámetro style=wx.TE_PROCESS_ENTER para que este sea capaz de procesar el pulsado de la tecla enter.

En cada uno de estos métodos se modifica a su vez la etiqueta usando el metodoGetValue() para tomar el valor actual del campo de texto y SetLabel() para establecer el nuevo valor de la etiqueta mas una leyenda discrecional con la acción realizada.

```
#!/usr/bin/python
try:
    import wx
except ImportError:
    raise ImportError,"Se requiere el módulo wxPython"

class simpleapp_wx(wx.Frame):
    def __init__(self,parent,id,title):
        wx.Frame.__init__(self,parent,id,title)
        self.parent = parent         self.initialize()

    def initialize(self):
        grilla = wx.GridBagSizer()
        self.entrada  =  wx.TextCtrl(self,-1,value=u"Ingrese  un  texto:",
style=wx.TE_PROCESS_ENTER)
        grilla.Add(self.entrada,(0,0),(1,1),wx.EXPAND)
       self.entrada.Bind(wx.EVT_TEXT_ENTER, self.SiPulsaEnter)
        boton = wx.Button(self,-1,label="Pulsa aquí")
        grilla.Add(boton, (0,1))
        boton.Bind (wx.EVT_BUTTON, self.SiCliqueaBoton)
        self.etiqueta = wx.StaticText(self,-1,label=u'Hola !')
        self.etiqueta.SetBackgroundColour(wx.BLUE)
        self.etiqueta.SetForegroundColour(wx.WHITE)
        grilla.Add(self.etiqueta, (1,0),(1,2), wx.EXPAND )
        grilla.AddGrowableCol(0)
        self.SetSizerAndFit(grilla)
        self.SetSizeHints(-1,self.GetSize().y,-1,self.GetSize().y )
        self.Show(True)

    def SiCliqueaBoton(self,event):
          self.etiqueta.SetLabel(self.entrada.GetValue() +  "Presionó el
botón!")

    def SiPulsaEnter(self,event):
```

```
        self.etiqueta.SetLabel(self.entrada.GetValue() + "Pulso enter!")


if __name__ == "__main__":
    app = wx.App()
    frame = simpleapp_wx(None,-1,'Mi aplicación')
    app.MainLoop()
```

Fuentes:

Interfaces gráficas de usuario en Python.

http://www.somoslibres.org/modules.php?name=News&file=article&sid=2226

Tutorial: Creating GUI Applications in Python with QT

http://www.cs.usfca.edu/~afedosov/qttut/

Curso de Interfaces Gráficas con Python [TkInter]

http://foro.el-hacker.com/index.php/topic,169236.0.html

Introducción a PyGTK y Glade

http://pythonmania.wordpress.com/2009/02/05/introduccion-a-pygtk-y-glade/

Escribir programas con wxPython en 15 pasos

http://www.retronet.com.ar/escribir-programas-con-wxpython-en-15-pasos/

Más referencias en la web

http://wiki.python.org/moin/TkInter

http://docs.python.org/library/tkinter.html

http://www.pygtk.org/

http://www.pygtk.org/pygtk2tutorial-es/index.html

http://pythonmania.wordpress.com/2009/02/05/introduccion-a-pygtk-y-glade/

http://wiki.python.org/moin/PyQt

http://www.cs.usfca.edu/~afedosov/qttut/

http://www.wxpython.org/

http://paginaspersonales.deusto.es/dipina/python/cursowxpython.ppt

http://www.retronet.com.ar/escribir-programas-con-wxpython-en-15-pasos/

http://foro.el-hacker.com/index.php/topic,169236.0.html

http://mundogeek.net/archivos/2008/11/24/interfaces-graficas-de-usuario-en- python/