



Tema 4 Interfaces gráficas con GTK

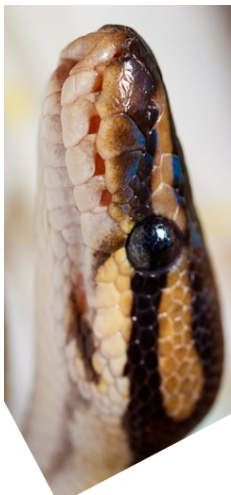
Curso de Python Avanzado

Juan Pedro Bolívar Puente

Instituto Andaluz de Astrofísica

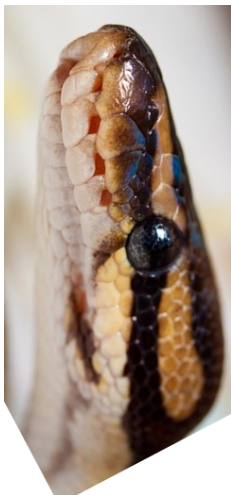
Mayo de 2011

Índice



- 1 Introducción
- 2 Fundamentos de GTK
- 3 Un paseo por los “widgets”
- 4 Conclusiones

Índice



- 1 **Introducción**
- 2 Fundamentos de GTK
- 3 Un paseo por los “widgets”
- 4 Conclusiones

Desarrolladores ...

¿Qué hemos visto hasta ahora?

Cosas de frikis...

- Lambdas
- Objetos
- Metaprogramación
- Estructuras infinitas



... y usuarios

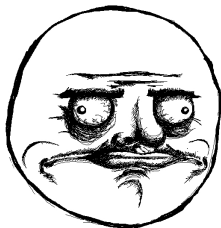
¿Qué quieren los
usuarios?

Hacer cosas
guays sin
pensar



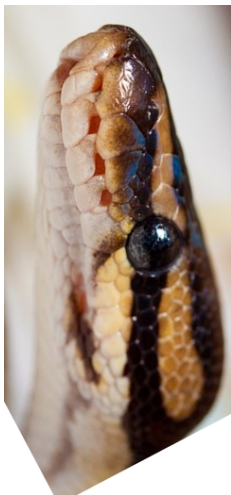
Las interfaces gráficas...

Las Interfaces Gráficas son la solución



- Usan **metáforas**
- Combinan **visualizar** y **modificar**
- Evitan **memorizar**
- Facilitan la **multitarea**

Índice



- 1 Introducción
- 2 Fundamentos de GTK
- 3 Un paseo por los “widgets”
- 4 Conclusiones

Frameworks de interfaces gráficas

Usamos un **framework** de interfaces gráficas

- **GTK**
Gnome, Ubuntu, Novell
- **Qt**
KDE, Kubuntu, Nokia
- **TkInter**
Python, Tcl/Tk
- **Interfaces nativas...**
¿WxWidgets?



Framework vs Biblioteca

Librería

Componentes que
llamamos desde
nuestro código

Framework

Estructura
que llama a
nuestro código

No nos llames, ya te llamaremos

EL PRINCIPIO DE HOLLYWOOD

La base de un programa en GTK

¡El main lo ejecuta GTK!

```
import pygtk
pygtk.require ('2.0')
import gtk

if __name__ == '__main__':
    gtk.main ()
```

Estructura del programa

- 1 Crear interfaz gráfica
- 2 Conectar manejadores
- 3 Ejecutar main



Eventos en GTK

La base de todo en GTK es `GObject`

- Un `GObject` expone **señales** — e.g. “`clicked`”
- En cada señal podemos **conectar** un **manejador**

```
def handler (objeto_emisor
             [, params fijados al emitir ]
             [, params fijados al conectar])
```

- Una señal se emite con ...

```
objeto_emisor.emit ("senal", ...)
```

Conectando a señales ...

Conectando señales

- `connect (sig, handler, ...)`
- `connect_after (sig, handler, ...)`

Re-envío de señales

- `connect_object (sig, handler, gobj)`
- `connect_object_after (sig, handler, gobj)`

Devuelen un `hdlid` que identifica la conexión

El grupo **after** se ejecuta después de las “normales”

Manejando la conexión

Desconectando la señal

- `disconnect (hdlid)`

¡Recuerda!

Una conexión
es un recurso

Bloqueando la señal

- `block (hdlid)`
- `unblock (hdlid)`

El bloqueo es
útil para evitar la
recursión infinita

Un ejemplo ...

Una clase con señales ...

```
import gobject
class MyClass (gobject.GObject):
    __gsignals__ = {
        'mysig' : (gobject.SIGNAL_RUN_FIRST,
                   None,
                   (object,))
    }
```

El diccionario `__gsignals__` permite definir las señales de los objetos de una clase

Un ejemplo ..

Usando nuestra clase ...

```
def my_handler (obj, param, *fixed):  
    print " _--_Handling_signal_--_"  
    print "Obj: _ _ _", obj  
    print "Param: _", param  
    print "Fixe: _ _", fixed  
  
obj = MyClass ()  
obj.connect ('mysig', my_handler, 'Hola!')  
  
obj.emit ('mysig', None)  
obj.emit ('mysig', 'param')
```


Las *propiedades*

Las **propiedades** son atributos “especiales” que emiten una señal al cambiar

No confundir con **property** de Python
No son necesariamente físicos, se manipulan internamente mediante una función

Las *propiedades*

Manipulando las propiedades ...

- `set_property[s]` (`property`, ...)
- `get_property[s]` (`property`, ...)

```
obj.props.width = 10  
var = obj.props.width
```

Requiere PyGTK 2.8

Controlando la emisión ...

- `notify` (`property_name`)
- `freeze_notify` ()
- `thaw_notify` ()

Ejemplo ...

Una clase con propiedades...

```
class MyClass (gobject.GObject):
    __gproperties__ = {
        'width' : (object,
                   'Ancho',
                   'El ancho del objeto',
                   gobject.PARAM_READWRITE)
    }

    def do_get_property (self, prop):
        return getattr (self, '_' + prop.name)

    def do_set_property (self, prop, val):
        setattr (self, '_' + prop.name, val)
```

Ejemplo ...

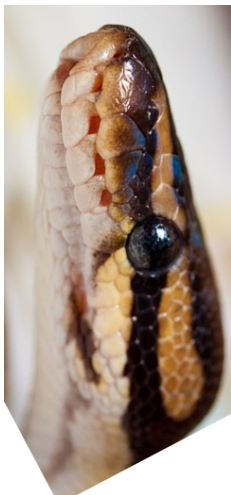
Usando las propiedades ...

```
a = MyClass ()

a.connect ('notify::width',
           my_handler)

a.props.width = 10
print a.props.width
```

Índice



- 1 Introducción
- 2 Fundamentos de GTK
- 3 Un paseo por los "widgets"
- 4 Conclusiones

¿Qué es un *widget*?

Widget = Window gadget

Un lenguaje gráfico universal

Ventanas, diálogos, botones, menús, pestañas, lista desplegable, caja de texto, etiquetas, opciones, multi-opciones, ...

Se organizan jerárquicamente

Una ventana **contiene** una pestaña que **contiene** una lista que **contiene** un botón que **contiene** una imagen ...

En GTK...

Todo lo que hereda de `gtk.Widget`

- Pueden estar **activados** o **desactivados**...
 - `show () / hide ()`
 - `show_all () / hide_all ()`
Lo aplica a **todo el subárbol** de la jerarquía
- Tiene una interfaz extensa común...
`[get,set]_name, get_parent, get_ancestor`

Los *oplevels*

oplevel = raiz del árbol de *widgets*
Derivados de `gtk.Window`

- GTK guarda una lista en `gtk.gdk.Screen`
- Al cerrarse una ventana **se destruye**
- Podemos evitar que se cierre devolviendo **False** en la señal "`delete-event`"

Gestión de recursos ...

Los objetos de GTK son un recurso

- Se "liberan" con `gtk.Object.destroy ()`
- Al hacerlo, emiten "`destroy`", que significa:
¡Borra cualquier referencia que tengas hacia mí!
- El objeto `sigue usable` si sigue habiendo referencias

En la **práctica** necesario sólo en *oplevels*

Recordemos al amigo with

Disponible en `gfits/src/util.py`

```
@contextmanager
def destroying (thing):
    try:
        yield thing
    finally:
        thing.destroy ()
```

Veremos su utilidad con los diálogos...

Nuestra primera ventana...

```
import gtk

win = gtk.Window ()
win.connect ('destroy', gtk.main_quit)
win.show ()

gtk.main ()
```

También podríamos parar el bucle
en la señal "delete-event"

Metiendo cosas en la ventana ...

¿Cómo **organizamos** los *widgets* dentro de la ventana?

Primera idea:

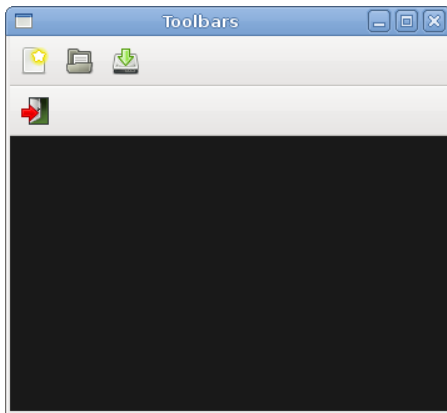
Colocarlos en posiciones (X, Y) determinadas

Mala idea:

Las ventanas deben poder cambiar de tamaño

Los contenedores ...

Usamos **contenedores** que proveen "huecos" que se deforman conforme ciertas restricciones



Disposición vertical y horizontal

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

- `pack_start (child, expand=True, fill=True, padding=0)`
Añade `child` al principio de la pila vertical u horizontal
- `pack_end (child, expand=True, fill=True, padding=0)`
Añade `child` al principio de la pila vertical u horizontal

Nuestro primer *widget* útil

```
gtk.Button (label=None, stock=None)
```

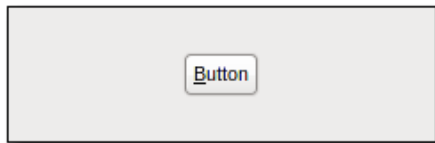
Lo usamos para desatar acciones

Señales importantes

- clicked
- pressed
- released
- enter
- leave

Propiedades importantes

- label
- image



Más widgets ...

```
gtk.Label (label=None, stock=None)
```

Etiquetas de texto

Señales importantes

- `activate-link`
- `copy-clipboard`



Label

Propiedades importantes

- `label`
- `use-markup`
- `ellipsize`
- `justify`
- `selectable`
- `wrap`

Al fin algo más serio...

```
win = gtk.Window ()
win.connect ('destroy', gtk.main_quit)

lab = gtk.Label ('Hola...')
btn = gtk.Button ('Pinchame')
btn.connect ('clicked', lambda *a:
              lab.set_text ('...mundo!'))

box = gtk.HBox ()
win.add (box)
box.pack_start (btn)
box.pack_start (lab)

win.show_all ()
gtk.main ()
```

El menu principal

Se construye con una jerarquía
`MenuBar` → `Menu` → `MenuItem`

- `MenuBar` es la barra de menu de la que cuelgan los...
- `Menu` son los paneles que se despliegan
- `MenuItem` son los elementos

Se usa `append (item)` para añadir `items` aun menu y `set_submenu (menu)` para establecer las jerarquías

Más widgets ...

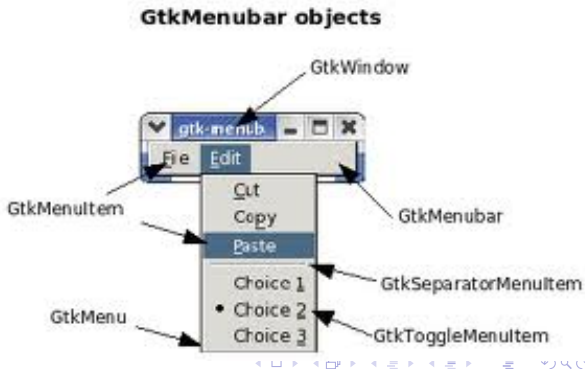
Lo más importante está en
`gtk.MenuItem` (`label=None`, `use_underline=True`)

Señales importantes

- `activate`

Propiedades importantes

- `submenu`
- `accel-path`



```
win = gtk.Window ()
win.connect ('destroy', gtk.main_quit)

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

menu_bar = gtk.MenuBar ()
box.pack_start (menu_bar, False)

file_menu = gtk.Menu ()
quit_item = gtk.MenuItem ('Quit')
file_menu.append (gtk.MenuItem ('Open'))
file_menu.append (gtk.MenuItem ('Save'))
```

```
quit_item = gtk.MenuItem ('Quit')
file_menu.append (quit_item)
quit_item.connect_object (
    'activate', gtk.Window.destroy, win)

file_item = gtk.MenuItem ('File')
file_item.set_submenu (file_menu)
menu_bar.append (file_item)

win.show_all ()
gtk.main ()
```

Esto se hace más fácil con [ItemFactory](#)

Otros contenedores básicos ...

`gtk.Toolbar ()`

Nuestra típica barra de herramientas ...

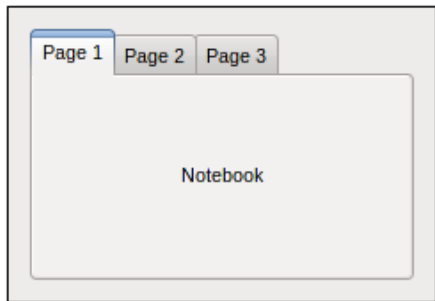
- `append_item (text, tooltip_text, tooltip_private_text, icon, callback, user_data=None)`
- `prepend_item (text, tooltip_text, tooltip_private_text, icon, callback, user_data=None)`



Todos quieren pestañas ...

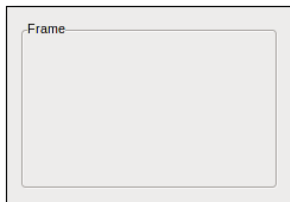
`gtk.Notebook ()` Un cuaderno con pestañas

- `append_page (child, label)`
- `prepend_page (child, label)`
- `insert_page (child, label, position)`
- `remove_page (page_num)`
- `get_current_page ()`

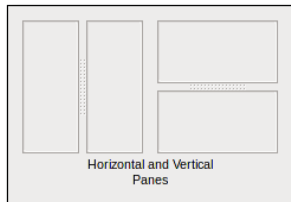


Y muchos más ...

- Fixed ()
- Layout ()
- Frame ()



- [H,V]ButtonBox ()
- [H,V]Paned ()
 - add1 ()
 - add2 ()




```
win = gtk.Window ()
win.connect ('destroy', gtk.main_quit)

toolbar = gtk.Toolbar ()

close_button = toolbar.append_item (
    "Open", "Open_a_file", "Private",
    gtk.image_new_from_stock(gtk.STOCK_OPEN,32),
    lambda *a: None)

close_button = toolbar.append_item (
    "Close", "Closes_this_app", "Private",
    gtk.image_new_from_stock(gtk.STOCK_CLOSE,32),
    lambda *a: win.destroy ())
```

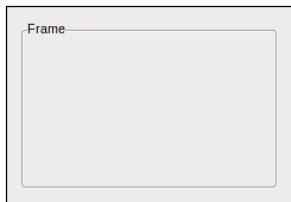
```
notebook = gtk.Notebook ()
notebook.append_page (gtk.Frame('Uno'),
                      gtk.Label('Uno'))
notebook.append_page (gtk.Frame('Dos'),
                      gtk.Label('Dos'))

box = gtk.VBox (False, 0)
box.pack_start (toolbar, False)
box.pack_start (notebook)

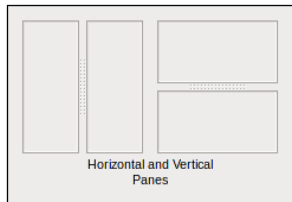
win.add (box)
win.show_all ()
gtk.main ()
```

Y muchos más ...

- Fixed ()
- Layout ()
- Frame ()



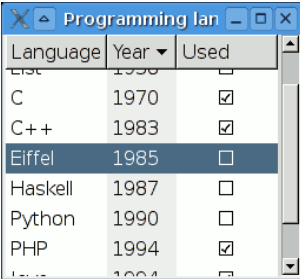
- [H,V]ButtonBox ()
- [H,V]Paned ()
 - add1 ()
 - add2 ()



Y muchos más ...

Las listas siguen una arquitectura modelo-vista-controlador

- **Modelo** Contiene la información "estática"
- **Vista** Representa el modelo, se entera si cambia por **señales**
- **Controlador** Cambia los valores del modelo, e.g. en respuesta al usuario



Language	Year	Used
Basic	1950	<input type="checkbox"/>
C	1970	<input checked="" type="checkbox"/>
C++	1983	<input checked="" type="checkbox"/>
Eiffel	1985	<input type="checkbox"/>
Haskell	1987	<input type="checkbox"/>
Python	1990	<input type="checkbox"/>
PHP	1994	<input checked="" type="checkbox"/>
Java	1994	<input type="checkbox"/>

El MVC de las listas

Modelo

Heredan de `TreeModel` `TreeStore`, `ListStore`,
`TreeModelSort`, `TreeModelFilter` ...

Vista

El `TreeView`

- Asocia `TreeViewColumn` a columnas del modelo
- Cada columna representa el contenido con un `CellRenderer(Pixbuf | Text | Toggle)`

Controlador

Los *manejadores* que manipulen el `TreeModel`

```
win = gtk.Window ()
win.connect ('destroy', gtk.main_quit)

model = gtk.ListStore (gobject.TYPE_BOOLEAN,
                       gobject.TYPE_STRING)

view = gtk.TreeView (model)
view.append_column (gtk.TreeViewColumn (
    'Done', gtk.CellRendererToggle (),
    active = 0))
view.append_column (gtk.TreeViewColumn (
    'Task', gtk.CellRendererText (),
    text = 1))
```

```
model.append ((True, 'Python_▯avanzado'))
model.append ((False, 'Interfaces_▯graficas'))
model.append ((False, 'Django'))

win.add (view)
win.show_all ()
gtk.main ()
```

Aunque el resto lo haremos con Glade, ¡las listas
hay que tocarlas a mano!

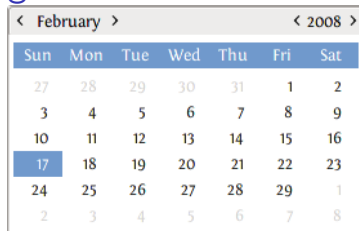
Otros widgets ...

Hay muchos *widgets* especializados que debéis consultar cuando necesitéis

gtk.TextView



gtk.Calendar



Integrando Matplotlib

Matplotlib se integra con GTK o con Qt

FigureCanvas

Es un `gtk.DrawingArea` que representa una figura

NavigationToolbar

Es la barra de herramientas que aparece con
`mplotlib.pyplot.show ()`

Hay varias versiones disponibles ...

```
# from matplotlib.backends.backend_gtk \
#     import FigureCanvasGTK as FigureCanvas
# from matplotlib.backends.backend_gtkcairo \
#     import FigureCanvasGTKCairo as FigureCanvas

from matplotlib.backends.backend_gtkagg \
    import FigureCanvasGTKAgg as FigureCanvas

# from matplotlib.backends.backend_gtk \
#     import NavigationToolbar2GTK as NavigationToolbar

from matplotlib.backends.backend_gtkagg import \
    NavigationToolbar2GTKAgg as NavigationToolbar

from matplotlib.figure import Figure
from numpy import arange, sin, pi
```

Creamos una ventana y una bonita senoide

```
win = gtk.Window ()
win.connect ("destroy", gtk.main_quit)
win.set_default_size (400, 300)
win.set_title ("Embedding in GTK")

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

fig = Figure (figsize=(5, 4), dpi=100)
ax = fig.add_subplot (111)
t = arange (0.0, 3.0, 0.01)
s = sin (2 * pi * t)
ax.plot (t, s)
```

¡Ta chán!

Y añadimos a la ventana ...

```
canvas = FigureCanvas (fig)
vbox.pack_start (canvas)

toolbar = NavigationToolbar (canvas, win)
vbox.pack_start (toolbar, False, False)

win.show_all ()
gtk.mainloop ()
```

¡Al actualizar la figura habría que llamar
`figure.canvas.draw ()`!

Diálogos

Diálogo = Ventana emergente que **bloquea la aplicación** hasta que le contestamos ...

Ejemplos ...

- FileChooserDialog
- AboutDialog
- ColorSelectionDialog
- FontSelectionDialog
- MessageDialog

Ejemplo ...

Tiene un método `run ()` que lo ejecuta y devuelve el resultado del usuario

¡Recordad que por ser una ventana hay
llamar a `destroy()`!

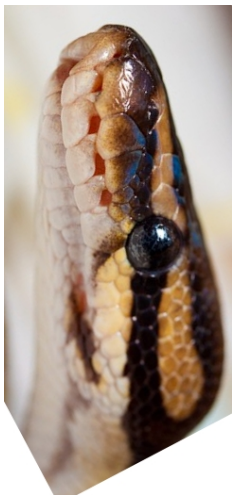
Una utilidad de regalo ...

De gfits/src/util.py

```
def show_message_dialog (msg, long_msg,
                          msg_type = gtk.MESSAGE_INFO):

    error_dlg = gtk.MessageDialog (
        type = msg_type,
        buttons = gtk.BUTTONS_CLOSE,
        message_format = long_msg)
    error_dlg.set_title (msg)
    res = error_dlg.run ()
    error_dlg.destroy ()
return res
```

Índice



- 1 Introducción
- 2 Fundamentos de GTK
- 3 Un paseo por los “widgets”
- 4 **Conclusiones**

Conclusiones



- La programación orientada a eventos nos obliga a usar señales
- Lo importante es qué ocurre no cuando ocurre
- Escribir interfaces a mano es pesado

Recursos adicionales



PyGTK Tutorial

John Finlay

[http:](http://www.pygtk.org/pygtk2tutorial/index.html)

[//www.pygtk.org/pygtk2tutorial/index.html](http://www.pygtk.org/pygtk2tutorial/index.html)



PyGTK Notebook, A Journey Through Python Gnome Technologies

Peter Gill

http://www.majorsilence.com/PyGTK_Book

¿Preguntas?

Muchas gracias por su atención.

