

PyGTK GUI programming

Table of Contents

PyGTK GUI programming.....	1
Chapter 1. Introduzione.....	2
1.1. Primo approccio.....	2
1.2. Il toolkit PyGTK.....	2
1.3. PyGTK e Glade.....	2
1.4. IDE o editor.....	4
1.5. Installazione.....	6
1.5.1. Installazione su piattaforma GNU/Linux.....	6
1.5.2. Installazione su piattaforma Windows.....	6
1.6. Supporto e help.....	6
Chapter 2. I Widget, le classi ed un primo esempio.....	8
2.1. I widget.....	8
2.2. Un primo esempio di GUI.....	8
2.3. Estendiamo l'esempio, un classico...Hello, World!.....	9
Chapter 3. Segnali, eventi e callbacks.....	11
3.1. I segnali e la classe padre gobject.GObject.....	11
3.1.1. Connettere un segnale.....	11
3.1.2. Connettere un segnale ad un oggetto.....	13
3.2. Gli eventi.....	14
3.2.1. Capire quale tasto e' stato premuto.....	14
Chapter 4. Gli attributi e le proprieta' dei widget.....	16
4.1. Gli attributi dei widget.....	16
4.2. Le proprieta' dei widget.....	16
Chapter 5. Impacchettare i widget.....	17
5.1. Utilizzare i packing box.....	17
5.1.1. Un po' di pratica con i packing box.....	17
5.2. Utilizzare la packing table.....	18
5.2.1. Un po' di pratica con la packing table.....	19
Chapter 6. Il widget contenitore, La finestra.....	20
6.1. gtk.Window().....	20
6.1.1. Associare e leggere un titolo dalla window.....	20
6.1.2. Decidere la posizione della window sullo schermo.....	21
6.1.3. I widget transitori.....	21
6.1.4. Distruggere la finestra con i widget transitori.....	22
6.1.5. Gestire il ridimensionamento della finestra.....	22
6.1.6. Muovere la finestra.....	22
6.1.7. Settare le decorazioni della finestra.....	23
6.1.8. Settare le dimensioni della finestra.....	24
6.1.9. Inserire un' icona nelle decorazioni della Window.....	24
6.1.10. Segnali ed eventi della Window.....	25
6.1.11. Le proprieta' della Window.....	27
6.1.12. Gli attributi della Window.....	30
Chapter 7. I bottoni.....	32
7.1. La gerarchia dei bottoni.....	32
7.2. gtk.Button().....	32

Table of Contents

Chapter 7. I bottoni	
7.2.1. Utilizzare gli Stock Item.....	34
7.2.2. Utilizzare i mnemonic accelerator.....	34
7.2.3. I segnali emessi da Button.....	34
7.2.4. Le proprieta' del Button.....	36
7.3. gtk.ToggleButton().....	37
7.3.1. Un primo esempio di ToggleButton.....	37
7.3.2. I segnali di ToggleButton.....	38
7.3.3. Le proprieta' del ToggleButton.....	38
7.3.4. Gli attributi del ToggleButton.....	39
7.4. gtk.CheckButton().....	39
7.4.1. Un primo esempio di CheckButton.....	39
7.4.2. Segnali, proprieta' ed attributi del CheckButton.....	40
7.5. gtk.RadioButton().....	40
7.5.1. Un primo esempio di RadioButton.....	40
7.5.2. Leggere e cambiare i gruppi di appartenenza.....	41
7.5.3. I segnali di RadioButton.....	42
7.5.4. Le proprieta' del RadioButton.....	42
7.5.5. Gli attributi del ToggleButton.....	42
Chapter 8. Metodi, attributi, proprieta' e segnali delle classi widget, container e bin.....	43
8.1. La classe widget.....	43
8.1.1. Impadronirsi del controllo.....	43
Chapter 9. Scendiamo di un livello, il wrapper GDK.....	45
9.1. gtk.gdk.Window().....	45
9.1.1. Catturare una gtk.gdk.Window.....	45
9.2. gtk.gdk.Cursor().....	46
9.2.1. Cambiare il cursore di una gtk.gdk.Window.....	46
9.3. gtk.gdk.Keymap().....	47
9.3.1. Catturare una gtk.gdk.Keymap.....	47
Chapter 10. Un ulteriore widget fondamentale, la Label.....	49
10.1. gtk.Label().....	49
10.1.1. Inserire il testo nella Label.....	50
10.1.2. Utilizzare i mnemonic accelerator.....	51
10.1.3. Associare un mnemonic accelerator ad un widget esterno.....	51
10.1.4. Leggere il testo associato ad una Label.....	52
10.1.5. Utilizzare il Pango markup in una Label.....	52
10.1.6. Operazioni di selezione in una Label.....	53
Chapter 11. Interagire con l'utente.....	56
11.1. gtk.Entry().....	56
11.1.1. Leggere e scrivere il testo.....	57
11.1.2. Inserire il testo nella Entry.....	58
11.1.3. Rendere invisibile il testo.....	59
11.1.4. Impostare il numero massimo di caratteri visualizzabili.....	60
11.1.5. Modificare l'allineamento del testo.....	61
Chapter 12. Messaggi di dialogo.....	62
12.1. gtk.Dialog().....	62
12.1.1. Un primo esempio di Dialog.....	62

Table of Contents

Chapter 12. Messaggi di dialogo	
12.2. gtk.MessageDialog()	63
12.2.1. Un primo esempio di MessageDialog	64
12.2.2. Utilizzare il markup nella Label all'interno di un MessageDialog	65
Chapter 13. I tool widget e la statusbar	67
13.1. gtk.Toolbar()	67
13.1.1. Inserire un item nella Toolbar	67
13.2. gtk.Tooltips()	68
13.2. gtk.Tooltips()	68
13.2.1. Abilitare o disabilitare un gruppo di Tooltips	68
13.2.2. Assegnare un Tooltips ad un widget	69
13.3. gtk.Statusbar()	70
Chapter 14. Manipolare il testo	71
14.1. gtk.TextView()	71
14.1.1. Un primo esempio di TextView	71
14.1.2. Associare e recuperare un TextBuffer dalla TextView	72
14.2. gtk.TextBuffer()	72
14.2.1. Contare linee e caratteri all'interno di un TextBuffer	73
14.3. gtk.TextIter()	74
14.3.1. Catturare il TextIter alla posizione corrente	74
14.4. gtk.TextMark()	75
14.4.1. Catturare il TextMark alla posizione corrente	75
14.5. gtk.TextTagTable()	76
14.6. gtk.TextTag()	76
Chapter 15. Look and feel	77
15.1. Modificare il colore di background di una Window	77
15.2. Modificare il colore di foreground di una Label	78
15.3. Modificare il colore di fg di una Label all'interno di un RadioButton	79
15.4. Creare un bottone con stock e testo personalizzato	81
Chapter 16. Riutilizzare il codice	82
16.1. Una finestra riutilizzabile	82
Appendix A. Recensione in 5 minuti	84
Appendix B. Tips and tricks	89
Appendix C. Lista degli esempi	91
Appendix D. Storia delle revisioni	94
Appendix E. A proposito del libro	95
Appendix F. GNU Free Documentation License	96
F.0. Preamble	96
F.1. Applicability and definitions	96
F.2. Verbatim copying	97
F.3. Copying in quantity	97
F.4. Modifications	97

Table of Contents

Appendix F. GNU Free Documentation License

F.5. Combining documents.....	99
F.6. Collections of documents.....	99
F.7. Aggregation with independent works.....	99
F.8. Translation.....	99
F.9. Termination.....	100
F.10. Future revisions of this license.....	100
F.11. How to use this License for your documents.....	100

Appendix G. Python license.....101

G.A. History of the software.....	101
G.B. Terms and conditions for accessing or otherwise using Python.....	101
G.B.1. PSF license agreement.....	101
G.B.2. BeOpen Python open source license agreement version 1.....	102
G.B.3. CNRI open source GPL-compatible license agreement.....	102
G.B.4. CWI permissions statement and disclaimer.....	103

PyGTK GUI programming

02 Gennaio 2005

Copyright © 2004, 2005 Gian Mario Tagliaretti (mailto:g.tagliaretti@parafernalìa.org)

Questo libro cresce di giorno in giorno, spero possa diventare un riferimento per chi vuole programmare GUI con Python e PyGTK

Permission is granted to copy, distribute, and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in Appendix F, *GNU Free Documentation License*.

The example programs in this book are free software; you can redistribute and/or modify them under the terms of the Python license as published by the Python Software Foundation. A copy of the license is included in Appendix G, *Python license*.

Chapter 1. Introduzione

1.1. Primo approccio

Sviluppare applicazioni in maniera rapida ed efficace e' ormai diventato un must, i tempi per la realizzazione dei progetti sono sempre piu' stretti, al programmatore viene richiesto tutto e subito, per queste ragioni l'accoppiata Python-PyGTK puo' essere una delle scelte migliori.

Molto di frequente navigando sui newsgroup e sui forum si sente criticare la lentezza dei linguaggi interpretati, dall'altro lato un'affermazione ricorrente e' che i processori hanno ormai raggiunto una potenza di calcolo che difficilmente puo' essere saturata con l'uso quotidiano di un computer.

Miscelando con cura le precedenti affermazioni si puo' forse trarne una terza:

Il coding di applicazioni basate su linguaggi interpretati e' diventato il giusto compromesso tra velocita' di sviluppo e velocita' di esecuzione, quello che Python e PyGTK vogliono offrire e' proprio questo.

1.2. Il toolkit PyGTK

Pygtk e' un set di moduli che permettono l'interfaccia tra Python e le librerie GTK, e' un toolkit object oriented, permette quindi la possibilita' di riusare lo stesso codice in piu' applicazioni, e' stato portato ai Python coders da James Henstridge in collaborazione con un team di developers volontari.

I programmi scritti con l'accoppiata Python-PyGTK sono generalmente corti, facili da leggere ed interpretare, inoltre applicazioni preesistenti scritte in Python possono generalmente essere riadattate con l'applicazione di una GUI (Graphical User Interface) con poche modifiche al codice originale.

Un forte sostenitore di pygtk e' da sempre red-hat, molti dei tool utilizzati nelle loro distribuzioni storiche, fino ad arrivare alla recente Fedora, sono stati scritti utilizzando questo toolkit, ad esempio anaconda il famosissimo installer e' stato scritto utilizzando proprio PyGTK.

Capita molto spesso di ritrovarsi a dover fare dei tentativi durante lo sviluppo di applicazioni complesse, grazie alla sintassi compatta di PyGTK fare delle prove con codice che probabilmente non verra' mai piu' utilizzato non diventa uno spreco di risorse insostenibile, la stessa cosa con programmi scritti in C/C++ ad esempio, non sarebbe possibile.

1.3. PyGTK e Glade

Glade, per chi non ne avesse mai sentito parlare, e' un utilissimo strumento che permette di organizzare una GUI in maniera visuale.

La GUI verra' creata utilizzando i widget presenti nella toolbar di glade, trascinando questi ultimi all'interno di una finestra principale ed organizzandoli secondo le proprie esigenze.

Glade puo' essere avviato con il supporto a gnome, cosi' facendo si avranno a disposizione widget aggiuntivi specifici per la piattaforma gnome.

L'interfaccia di glade si presenta con 4 finestre:

Figure 1.1. La finestra principale del programma

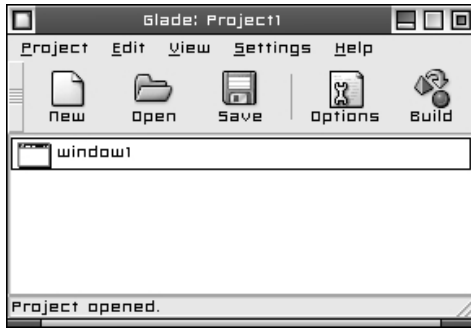


Figure 1.2. La toolbar contenente tutti i widget a disposizione

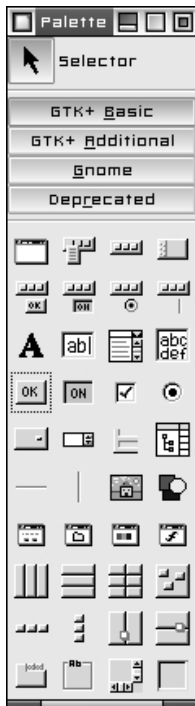


Figure 1.3. La finestra contenente le proprieta' dei widget

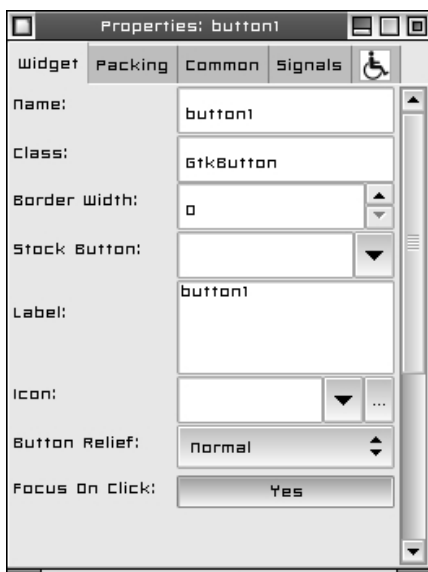
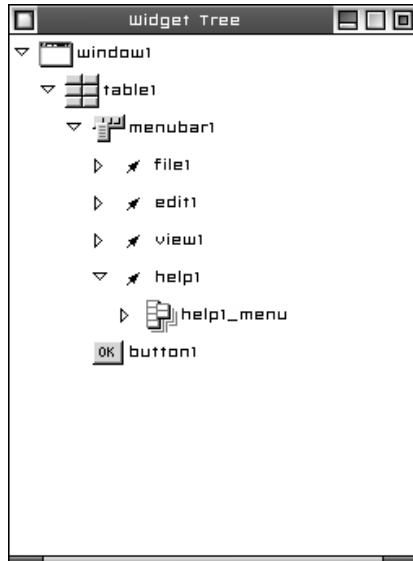


Figure 1.4. La vista ad albero dei widget

Glade non produce output in puro codice Python, ma sarà salvato in formato XML, successivamente tramite la libreria libglade potrà essere importato ed utilizzato dai nostri programmi Python, più avanti vedremo degli esempi approfonditi sull'utilizzo di Glade.

1.4. IDE o editor

Come per qualsiasi altro linguaggio, nello scrivere applicazioni con Python–PyGTK bisogna decidere quali tool di sviluppo utilizzare, generalmente per sviluppare semplici programmi un editor è più che sufficiente, mentre per applicazioni più complesse dove magari può rendersi necessaria una fase di debug, è meglio passare ad utilizzare una delle tante IDE (Integrated Development Environment) a disposizione.

La discussione su quale sia il miglior editor o quale sia la miglior IDE, sfocia spesso e volentieri in dei flame su entrambi, newsgroup e forum, il mio consiglio è quello di provare le più blasonate e fermarsi nella ricerca quando uno strumento sembra essere quello che meglio si adatta alle proprie esigenze, tenendo sempre presente che l'editor o la IDE perfetti non esistono.

Tra gli editor, quelli che vale la pena menzionare sono sicuramente gli intramontabili Emacs e Vim, DrPython, Scite, PyPE. Possono essere utilizzati anche i tools che generalmente vengono installati con i più famosi desktop environment, bluefish per Gnome e kate per KDE.

Sono molte le IDE disponibili, nel nostro caso specifico Python–PyGTK ma non solo, quelle da citare sono fondamentalmente quattro, le prime tre sono IDE che non dipendono dal desktop environment su cui vengono eseguite, Eclipse con il plugin pydev per il supporto a Python, Eric3 e Anjuta, la quarta IDE è Kdevelop che risiede sulla piattaforma KDE.

Quale scegliere è solo una questione di preferenza personale, qui di seguito alcuni screenshot

Figure 1.5. Eclipse

PyGTK GUI programming

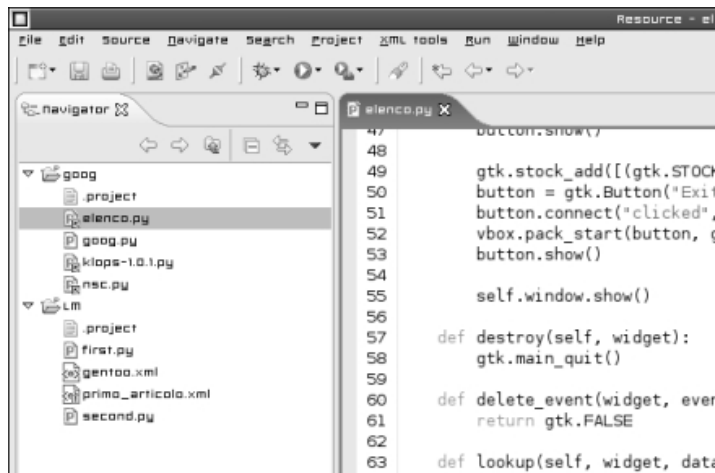


Figure 1.6. Eric 3

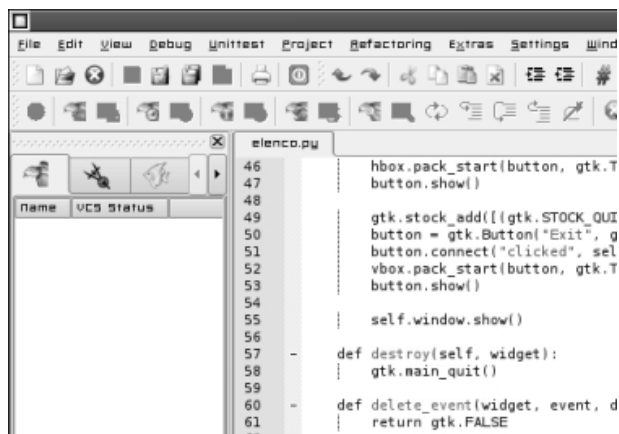


Figure 1.7. Kdevelop

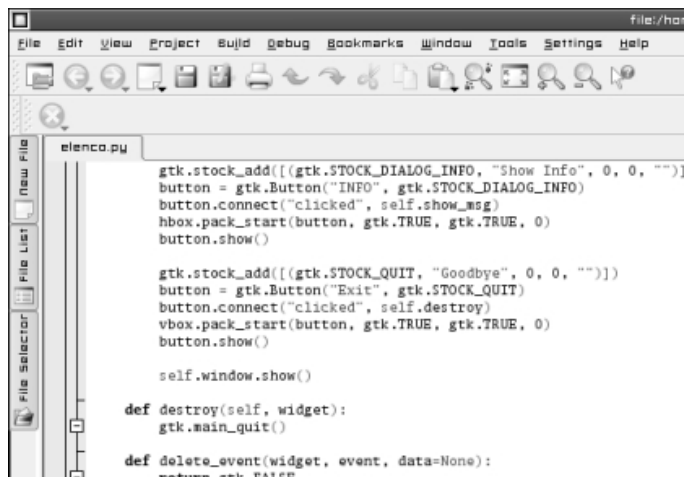
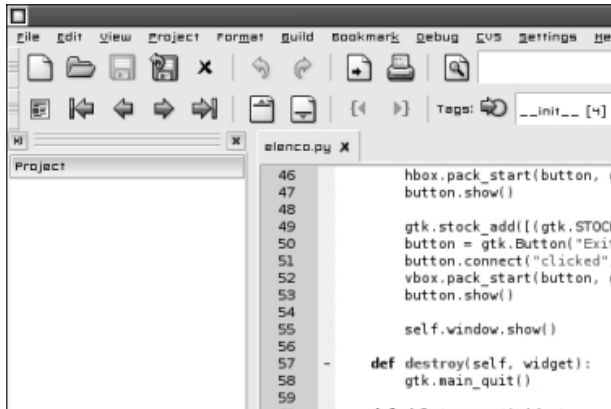


Figure 1.8. Anjuta



1.5. Installazione

PyGTK puo' essere utilizzato indipendentemente dalla piattaforma, e' pienamente supportato da GNU/Linux e Windows, puo' essere utilizzato anche sul Mac installando un Xserver.

1.5.1. Installazione su piattaforma GNU/Linux

Quasi tutte le maggiori distribuzioni GNU/Linux forniscono i package binari per l'installazione di PyGTK, nel caso si voglia installare da sorgenti i file possono essere scaricati dalla sezione download di www.pygtk.org (www.pygtk.org).

Installando da sorgenti bisogna prestare molta attenzione alle dipendenze da rispettare, queste ultime dipendono molto dalla distribuzione GNU/Linux su cui si cerca di installare il toolkit.

1.5.2. Installazione su piattaforma Windows

Per poter utilizzare pygkt su macchine windows bisogna installare i seguenti pacchetti:

L'ultima release delle GTK per windows che potete scaricare da <http://www.dropline.net/gtk/download.php> (<http://www.dropline.net/gtk/download.php>)

Dopo aver installato il pacchetto bisogna settare le variabili d'ambiente modificando il PATH;

Andranno aggiunte le directory riguardanti le librerie e bin.

```

c:\Programmi\File Comuni\GTK\2.0\bin
c:\Programmi\File Comuni\GTK\2.0\lib

```

Infine andra' installato il pacchetto http://www.pcpm.ucl.ac.be/~gustin/win32_ports/

In aggiunta, se si volessero compilare le applicazioni, in modo da poter distribuire un pacchetto che possa girare su tutte le macchina windows come 'stand-alone', andra' installato anche il pacchetto py2exe (<http://starship.python.net/crew/theller/py2exe/>)

Piu' avanti ci sara' anche un breve capitolo riguardante la compilazione con py2exe di applicazioni PyGTK

1.6. Supporto e help

La documentazione disponibile sulla rete e' quasi esclusivamente in inglese, sono disponibili sia per la consultazione online sia per il download un tutorial ed un reference manual, entrambi redatti da John Finlay alla home page www.pygtk.org.

PyGTK GUI programming

Esiste un'ampia collezione di FAQ mantenuta aggiornata da Christian Reis, disponibile per la consultazione al sito <http://www.async.com.br/faq/pygtk/>

Un altro modo per avere supporto e' aderire alla mailing list ufficiale, per iscriversi il sito di riferimento e' www.daa.com.au/mailman/listinfo/pygtk la ML e' consultabile tramite gmane all' url <http://news.gmane.org/gmane.comp.gnome.gtk+.python/>

Sono inoltre disponibili un canale IRC #pygtk su irc.gnome.org.

Infine per riportare un bug o per cercare tra quelli inviati, il link di riferimento e' <http://bugzilla.gnome.org/query.cgi>

Chapter 2. I Widget, le classi ed un primo esempio

2.1. I widget

Come già precedentemente menzionato, il toolkit PyGTK è Object Oriented, i widget non sono altro che delle classi con dei metodi associati.

L'organizzazione di queste classi è ad albero, come vedremo più avanti tutti i componenti del toolkit discendono da GObject che è la classe padre. I widget non discendono direttamente da quest'ultima ma da una classe intermedia, come possiamo vedere dalla gerarchia:

La gerarchia

```
+-- GObject
+-- gtk.Object
+-- gtk.Widget
```

Tutti i metodi, gli attributi e le proprietà di questa classe vengono ereditate da tutti i widget che da essa discendono, tranne delle eccezioni che vedremo strada facendo.

Durante tutto il percorso in grassetto sarà sempre indicata la classe che stiamo prendendo in esame.

2.2. Un primo esempio di GUI

Passiamo ora ai fatti creando la prima finestra con PyGTK!!

Come già precedentemente menzionato, nonostante sia possibile scrivere codice procedurale l'approccio migliore, anche in termini di riuso del codice, è quello di scrivere applicazioni utilizzando le tecniche OOP, utilizzeremo quindi le classi anche per semplici programmi.

Example 2.1. La creazione di una Window

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

class FirstWin:
    def __init__(self):
        self.win = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.win.show()

    def main(self):
        gtk.main()

if __name__ == "__main__":
    first = FirstWin()
    first.main()
```

first_window.py (../pyhtml/first_window.html)

Nonostante siano solo poche righe di codice, questo primo programma presenta già una serie di contenuti interessanti:

Import del modulo pygtk

PyGTK GUI programming

controllare che la versione presente nel sistema e su cui dovra' girare l'applicazione, deve essere della serie 2.0 (o superiore), questo e' un semplice ed utile trucco per evitare conflitti di versione, potremmo infatti avere piu' versioni di PyGTK installate su di uno stesso pc.

Successivamente verra' caricato il modulo gtk contenente i widget che andremo ad utilizzare nello sviluppo della nostra GUI, al caricamento della libreria verra' automaticamente eseguita la funzione `gtk_init()` la quale, senza scendere in troppi dettagli, si occupera' di inizializzare e preparare il toolkit dall' uso.

La dichiarazione della classe `FirstWin`

e' la volta del costruttore `__init__()`, che nel nostro caso non conterra' alcun parametro se non `self`

La dichiarazione di una nuova `gtk.Window`, questo e' il widget che generalmente viene utilizzato per fare da contenitore a tutti gli altri widget che saranno aggiunti in seguito, per default la finestra creata sara' di 200x200 pixel, vedremo negli esempi successivi come modificare questa ed altre impostazioni predefinite. Il widget `Window` e' inoltre corredato dalla top bar contenente i pulsanti che ne permetteranno la chiusura, il ridimensionamento e lo spostamento, anche questo attributo, `decorated()` potra' essere modificato, come stiamo per vedere.

Con l'istruzione successiva `win.show()` viene chiesto al window manager di mettere la finestra in display, questa istruzione come vedremo negli esempi successivi, andra' applicata a tutti i widget creati.

L'ultimo passo e' quello di creare la funzione `main()` che si occupera' di eseguire `gtk.mainloop()`.

Come sappiamo questo metodo permettere di eseguire il modulo, ovvero se non importato da altri moduli

A questo punto eseguendo il programma da un qualsiasi `*term`, apparira' una finestra simile a questa:

Figure 2.1. La prima finestra



Ovviamente il look della finestra dipendera' dal window manger che state utilizzando e dalle sue impostazioni, tutti gli esempi che vedrete sono fatti su di una Linuxbox su cui gira Gentoo GNU/Linux e come window manager fluxbox 0.9.11 oppure openbox 3.2-r1.

2.3. Estendiamo l'esempio, un classico...Hello, World!

Tutti gli esempi che si rispettino hanno un bottone Hello, World, potremmo essere da meno...??

Example 2.2. Un bottone Hello, World!

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

class FirstWin:
    def __init__(self):
        self.win = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.button = gtk.Button("Hello, World!")
```

PyGTK GUI programming

```
self.win.add(self.button)
self.win.show_all()

def main(self):
    gtk.main()

if __name__ == "__main__":
    first = FirstWin()
    first.main()
```

second_example.py (../pyhtml/second_example.html)

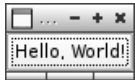
Il costruttore di un nuovo bottone, come argomento e' passato il testo che comparira' nella Label.

Il nuovo bottone e' quindi inserito nella finestra tramite il metodo add che analizzeremo in seguito.

Il metodo show_all e' una variante che permette con una sola dichiarazione di mostrare tutti gli oggetti contenuti nella Window.

Anche in questo caso eseguendo il programma da un qualsiasi *term, apparira' una finestra simile a questa:

Figure 2.2. Secondo esempio, finalmente un bottone!



Chapter 3. Segnali, eventi e callbacks

Nel primo esempio di Window non avendo ancora implementato una funzione che dica al window manager di chiudere la finestra ed uscire da `gtk.mainloop()`, alla pressione del tasto di chiusura nella top bar, la finestra sarà nascosta ma il programma continuerà a rimanere in esecuzione, per chiuderlo dovremo premere la combinazione di tasti `ctrl+c` sul *term attivo.

Prima di procedere all'implementazione di nuovi widget e nuove funzioni, è importante capire come funzionano segnali e callbacks del toolkit GTK+.

A partire dalla versione 2 delle librerie GTK+, la gestione di segnali e callback è stata spostata dal toolkit stesso alla libreria Glib, questo all'atto pratico non comporta nessuna modifica alla stesura del codice per chi aveva già esperienza con il ramo 1.x della libreria, in quanto l'operazione rimane totalmente trasparente al programmatore.

Come la maggior parte, se non tutti, i toolkit per la progettazione di GUI, anche PyGTK è di tipo event-driven, il che significa che il programma rimarrà in attesa di interagire con l'utente, questo avviene in quanto la funzione `gtk.main()` rimane in attesa, quando un evento viene rilevato, il controllo di quest'ultimo sarà passato alla funzione appropriata.

Questo passaggio del controllo sugli eventi è effettuato mediante dei segnali, questi ultimi non sono confondersi con i segnali di sistema UNIX nonostante la terminologia sia simile, quando un evento viene rilevato, come ad esempio la pressione di un bottone del mouse, il widget interessato emetterà un segnale.

Per permettere agli oggetti di interagire con l'utente si deve quindi utilizzare un metodo che catturi il segnale emesso dal widget, quest'ultimo deve appartenere alla classe padre `gobject.GObject`

3.1. I segnali e la classe padre `gobject.GObject`

Chi si occupa di connettere i segnali e passarli alla funzione appropriata è `gobject.GObject` con i suoi metodi.

Come è facilmente intuibile dalla gerarchia riportata poco più in basso, `gobject.GObject` è la classe padre di tutti i widget che compongono il toolkit GTK+

La gerarchia

```
+-- gobject.GObject
```

Come è facilmente intuibile dalla gerarchia `gobject.GObject` è la classe padre di tutti i widget che compongono il toolkit GTK+

3.1.1. Connettere un segnale

```
gobject.GObject.connect(signal, handler, *args)
```

signal: Una stringa contenente il nome del segnale

handler: Una funzione python o un metodo dell'oggetto

***args:** Argomenti supplementari

il metodo `gobject.GObject.connect()` riceve il segnale da un widget, ad esempio alla pressione di un Button, passa quindi il controllo alla funzione specificata da `handler`, `*args` sono gli argomenti supplementari da

passare alla funzione handler.

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

class SecondWin:
    def __init__(self):
        self.win = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.win.connect("delete_event", self.delete_event)
        self.win.connect("destroy", self.destroy)
        self.win.set_resizable(gtk.FALSE)
        self.win.set_title('wow un bottone!')
        self.win.set_border_width(5)
        self.vbox = gtk.VBox()
        self.win.add(self.vbox)
        self.button = gtk.Button("click me")
        self.button.connect("clicked", self.click, "Now I'm quit")
        self.vbox.pack_start(self.button)
        self.button1 = gtk.Button("Change me!")
        self.button1.connect("clicked", self.destroy)
        self.vbox.pack_start(self.button1)
        self.win.show_all()

    def click(self, widget, data=None):
        self.button1.set_label(data)

    def delete_event(self, widget, event, data=None):
        return gtk.FALSE

    def destroy(self, widget, data=None):
        return gtk.main_quit()

    def main(self):
        gtk.main()

if __name__ == "__main__":
    second = SecondWin()
    second.main()
```

button_connect_data.py (../pyhtml/button_connect_data.html)

Creazione del primo bottone di cui catturare la pressione tramite il click del mouse.

Cattura della pressione del tasto, "clicked" e' il nome del segnale da catturare, self.click la funzione di handler ed infine il testo da passare come argomento.

Creazione del secondo bottone che sara' l'oggetto della funzione self.click e di cui catturare la pressione tramite il click del mouse.

In questo caso alla pressione del Button sara' emesso il segnale destroy() tramite l'apposita funzione.

Questa e' la prima funzione di handler, eseguita in seguito alla pressione del primo Button.

il parametro data assumerà il valore "Now I'm quit" passato come argomento alla funzione e cambierà la label del secondo Button.

funzione che esegue il gtk.main_quit, il programma sara' terminato.

Riassumendo brevemente, cliccando sul primo Button il segnale "clicked" verra' emesso, il segnale sara' catturato ed eseguita la funzione handler, (self.click), il parametro aggiuntivo ovvero il nuovo testo per il secondo Button sara' passato come argomento, la label del secondo Button cambiata di conseguenza. Alla pressione di quest'ultimo quindi la funzione destroy() chiudera' la nostra applicazione.

All'esecuzione del programma la GUI si presentera' in questo modo:



In seguito alla pressione del tasto:



Cliccando sul bottone "Now I'm quit" usciremo dal programma.

3.1.2. Connettere un segnale ad un oggetto

Questo metodo puo' essere leggermente fuorviante all'inizio e potrebbe anche sembrare un duplicato del precedente, in realta' l'uso che se ne puo' fare e' estremamente utile, vediamo come sempre il prototipo ed un esempio pratico:

```
gobject.GObject.connect_object(signal, handler, gobject)
```

signal: Una stringa contenente il nome del segnale

handler: Una funzione python o un metodo dell'oggetto

gobject: Un GObject

il metodo **gobject.GObject.connect_object()** riceve il segnale da un widget, ad esempio alla pressione di un Button, passa quindi il controllo alla funzione specificata da handler, object e' un GObject quale puo' essere ad esempio una Window che si vuole chiudere alla pressione del suddetto Button.

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

class SecondWin:
    def __init__(self):
        self.win = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.win.connect("delete_event", self.delete_event)
        self.win.connect("destroy", self.destroy)
        self.win.set_resizable(gtk.FALSE)
        self.win.set_title('wow un bottone!')
        self.win.set_border_width(5)
        self.button = gtk.Button("Hide window!")
        self.win1 = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.win1.set_default_size(200, 200)
        self.button.connect_object("clicked", gtk.Widget.hide, self.win1)
        self.win.add(self.button)
        self.win.show_all()
        self.win1.show()

    def delete_event(self, widget, event, data=None):
        return gtk.FALSE

    def destroy(self, widget, data=None):
        return gtk.main_quit()

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

```
if __name__ == "__main__":
    second = SecondWin()
    second.main()
```

button_connect_object.py (../pyhtml/button_connect_object.html)

Alla pressione del Button il segnale sara' passato alla funzione `hide()` che avra' come oggetto la finestra `win1`.

Sfruttando questo metodo possiamo quindi interagire con widget diversi da quello a cui il segnale e' associato, molto comodo vero?

3.2. Gli eventi

Oltre alla gestione dei segnali esiste anche la gestione degli eventi, la differenza fondamentale tra i due e' che i primi (segnali) sono emessi dai widget, i secondi sono emessi dal server X e dalle periferiche ad esso collegate.

La gestione degli eventi e' comunque simile alla gestione dei segnali, l'unica differenza e' che la funzione di callback avra' un parametro aggiuntivo, per l'appunto **event**

3.2.1. Capire quale tasto e' stato premuto.

Durante lo sviluppo di una GUI a volte e' essenziale capire quale tasto sia stato premuto dallo user, in modo da poter decidere come la GUI si dovra' comportare, ad esempio a seguito della pressione del tasto `enter` piuttosto che alla pressione di una freccia direzionale. Per farlo dobbiamo ricorrere a due strumenti, per l'esattezza `gtk.gdk.Keymap` e la gestione degli eventi.

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

class FirstWin(gtk.Window):
    def __init__(self):
        gtk.Window.__init__(self, gtk.WINDOW_TOPLEVEL)
        self.set_default_size(100,100)
        self.connect("destroy", lambda w: gtk.main_quit())
        self.connect("key_press_event", self.doKeyPress)
        self.show()

    def doKeyPress(self, widget, event):
        keyname = gtk.gdk.keyval_name(event.keyval)
        print "the button %s was pressed" % keyname

    def main(self):
        gtk.main()

if __name__ == "__main__":
    first = FirstWin()
    first.main()
```

event_get_key.py (../pyhtml/event_get_key.html)

Metodo alternativo di connettere il segnale `destroy`, se la funzione `lambda` verra' eliminata in futuro e' meglio non utilizzarla, ma...e' comoda.

Cattura della pressione di un qualunque tasto tramite l'evento **key_press_event**, il controllo sara' passato alla finzione `doKeyPress`.

PyGTK GUI programming

Definiamo la funzione di callback, vediamo che e' presente il parametro supplementare **event**.

Tramite il metodo **gtk.gdk.keyval_name(event.keyval)** raccogliamo il nome del tasto premuto e lo associamo alla variabile **keyname**.

stampiamo infine sul `*term` il nome del tasto premuto.

Riassumendo brevemente, alla pressione di un qualunque tasto l'evento (**key_press_event**) sara' catturato dalla funzione di callback **doKeyPress**, il controllo sara' passato a quest'ultima che tramite uno dei metodi di `gtk.gdk.Keymap` ovvero **gtk.gdk.keyval_name** restituira' il nome del tasto premuto il quale sara stampato a video.

Chapter 4. Gli attributi e le proprieta' dei widget

Tutti i widget che compongono il toolkit PyGTK sono dotati di attributi e di proprieta'. La differenza fondamentale tra i due consiste nel metodo con cui accedervi.

4.1. Gli attributi dei widget

Gli attributi dei widget sono fondamentalmente delle caratteristiche degli stessi e per lo piu' sono di sola lettura.

Gli attributi sono un mapping alla struttura di GObject ed e' quindi possibile accedervi direttamente. Prendiamo ad esempio una Window a cui sia stato precedentemente assegnato un titolo, per accedere a quest'ultimo e' sufficiente fare:

```
titolo = window.title
```

Per il momento non ci addentriamo nella struttura del codice, nei prossimi capitoli vedremo per ogni singolo widget come utilizzare gli attributi.

4.2. Le proprieta' dei widget

Le proprieta' dei widget appartengono all'oggetto GObject, e' possibile accedervi tramite dei metodi di quest'ultimo.

A differenza degli attributi, le proprieta' dei widget non sono di sola lettura, nel 99% dei casi possono essere lette e scritte. Per farlo possiamo utilizzare due dei metodi di gobject.GObject:

```
gobject.GObject.get_property(property_name)
gobject.GObject.set_property(property_name, value)
```

Sempre prendendo ad esempio una Window, fanno parte delle sue proprieta' il tipo, l'altezza, la larghezza ecc...

Anche il titolo della Window fa parte delle proprieta' oltre che degli attributi, la differenza e' che come proprieta' ha la possibilita' di essere scritto, come attributo e' in sola lettura.

A volte e' piu' comodo settare le proprieta' dei widget tramite i loro attributi piuttosto che utilizzarne i metodi, come fa ad esempio Glade, sempre prendendo ad esempio la Window potremo cambiare il titolo sia modificando la proprieta' sia con il metodo dedicato.

Chapter 5. Impacchettare i widget

In fase di progettazione di un nuovo software avremo sicuramente bisogno di inserire piu' widget all'interno di una finestra padre, fino ad ora negli esempi e' sempre stato utilizzato un solo elemento, ma cosa succede quando i componenti diventano piu' di uno? Qui entra in gioco l' impacchettamento dei widget.

Ci sono 2 modi per distribuire widget all'interno di un contenitore, il primo e' inserirli allo interno di packing boxes l'alternativa e' utilizzare una table suddivisa in righe e colonne, in sostanza una griglia.

Entrambi i metodi sono efficaci, quale dei due utilizzare e' per lo piu' una scelta personale del progettista software, personalmente dopo aver provato entrambe le soluzioni ho fatto ricadere la mia scelta sui packing box che trovo molto piu' flessibili, ma puo' a volte capitare soprattutto per GUI molto lineari, di ritrovarsi ad utilizzare la griglia.

5.1. Utilizzare i packing box

I packing box non sono altro che una serie di scatole invisibili in cui inserire i widget, possiamo utilizzare 2 tipi differenti di box, orizzontali `gtk.HBox()` e verticale `gtk.VBox`, entrambi discendono dal widget padre `gtk.Box`.

Queste scatole invisibili possono essere inserite una dentro l'altra per ottenere l'effetto desiderato in fase di posizionamento dei widget. Per inserire gli elementi desiderati all'interno delle scatole si possono usare i due metodi messi a disposizione dal toolkit, ovvero `pack_start()` e `pack_end()`, il primo posizionera' gli oggetti da sinistra verso destra per `gtk.HBox()` e dall'alto verso il basso per `gtk.VBox()`, il secondo metodo chiaramente li sistemera' nel verso opposto, negli esempi che seguiranno sara' usato quasi esclusivamente il metodo `pack_start()`.

Utilizzando questi metodi GTK sapra' esattamente come posizionare i widget ed inoltre fara' il ridimensionamento automatico ed altre operazioni per noi, operazioni queste che potranno essere modificate per allineare il posizionamento alle nostre scelte. Come potete immaginare, questo e' uno strumento molto flessibile, anche se le prime volte che lo si utilizza puo' essere di non facile comprensione.

La gerarchia

```
+-- GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.Box
          +-- gtk.VBox
          +-- gtk.HBox
```

5.1.1. Un po' di pratica con i packing box

Passiamo ora a descrivere l'uso dei widget `gtk.Box`, innanzitutto il costruttore:

```
gtk.Box(homogeneous=FALSE, spacing=0)
```

Ovviamente `gtk.Box` andra' sostituito con `gtk.VBox` oppure `gtk.HBox`

Il parametro `homogeneous` determina se lo spazio allocato ai widget all'interno del box dovra' essere omogeneo, non nelle due direzioni, ma in verticale per `gtk.VBox` ed ovviamente in orizzontale per `gtk.HBox`. Il parametro `spacing` determina lo spazio aggiuntivo in pixel da applicare tra un widget e i suoi vicini all'interno dei `gtk.Box`.

PyGTK GUI programming

Con la funzione `pack_start()` o `pack_end()` i widget saranno inseriti nei `gtk.Box`, il prototipo del metodo valido per gli oggetti `gtk.Box` e' nella forma:

```
def pack_start(child, expand=TRUE, fill=TRUE, padding=0)
```

child: il widget da inserire in `gtk.Box`

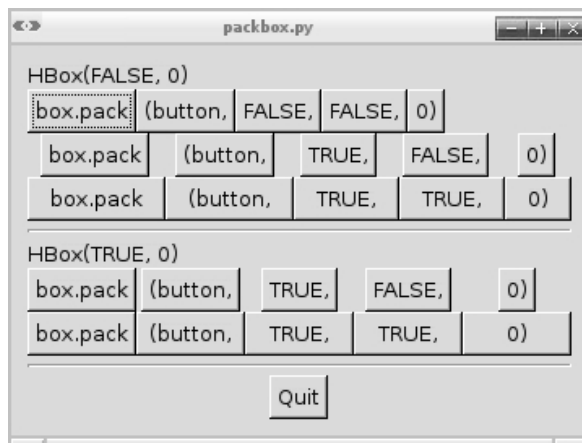
expand: se TRUE, il widget riceverà spazio aggiuntivo all'interno del box, pari al widget più grande in esso contenuto.

fill: se TRUE, il widget si espanderà in modo da avere uguale dimensione al widget più grande

padding: e' lo spazio in pixel tra il widget e i suoi vicini, questo spazio si sommerà a quello eventualmente già allocato quando e' stato costruito il box.

Questa e' una delle parti più difficili da comprendere e da spiegare, l'immagine qui sotto raccoglie tutte le possibili combinazioni, che in tutto sono cinque.

Figure 5.1. Esempio di Packing



Le combinazioni che possiamo ottenere sono cinque e non sei come si potrebbe in un primo momento pensare, il motivo e' molto semplice, se il parametro `homogeneous` del contenitore `gtk.HBox` viene impostato a TRUE, anche se impostassimo il parametro `expand` a FALSE inserendo i widget con `pack_start()`, quest'ultimo sarebbe ignorato.

Il modo migliore per imparare ad utilizzare i packing box rimane quello di fare molte prove, inizialmente scrivendo un codice che faccia da base e su di esso tentare tutte le possibili combinazioni, vedremo comunque molti esempi pratici di utilizzo di packing box.

5.2. Utilizzare la packing table

Un altro metodo per posizionare gli oggetti all'interno della GUI e' quello di usare una tabella, questo strumento a volte puo' essere davvero indispensabile,

Gli oggetti saranno disposti all'interno di una griglia che divide la nostra tabella, il numero di righe e colonne viene deciso in fase di creazione della `gtk.Table()`, i widget al suo interno possono occupare una o più righe e/o colonne.

```
    0           1           2           3
0 +-----+-----+-----+
  |         |         |         |
```

```

1 +-----+-----+-----+
  |         |         |         |
2 +-----+-----+-----+

```

e' da notare che il punto di partenza della griglia e' sempre in alto a sinistra.

La gerarchia

```

+-- GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.Table

```

5.2.1. Un po' di pratica con la packing table

L'uso di `gtk.Table()` e' concettualmente molto diverso, ecco il costruttore:

```
gtk.Table(rows=1, columns=1, homogeneous=FALSE)
```

Rows e **columns** corrispondono al numero di righe e colonne che comporranno la tabella. L'argomento **homogeneous** se impostato a `TRUE`, ridimensionera' le celle prendendo come dimensioni quelle del widget piu' grande nella tabella, se impostato a `FALSE` l'altezza delle celle sara' dettata dall'altezza del widget piu' alto nella stessa riga, la larghezza sara' determinata dal widget piu' largo presente nella stessa colonna.

La funzione di inserimento dei widget nelle tabelle e' `attach()`, ecco il suo prototipo:

```
def attach(child, left_attach, right_attach, top_attach, bottom_attach, xopt=gtk.EXPAND|gtk.FILL)
```

child: il widget da inserire in `gtk.Box`.

left_attach: il numero di colonna a cui attaccare il lato sinistro del widget.

right_attach: il numero di colonna a cui attaccare il lato destro del widget.

top_attach: il numero di riga a cui attaccare il lato alto del widget.

bottom_attach: il numero di riga a cui attaccare il lato basso del widget.

xoptions: usato per specificare la proprieta' del widget quando la tabella e' ridimensionata orizzontalmente.

yoptions: usato per specificare la proprieta' del widget quando la tabella e' ridimensionata verticalmente.

xpadding: numero di pixel da aggiungere a destra e a sinistra del widget.

ypadding: numero di pixel da aggiungere in alto e in basso al widget.

Chapter 6. Il widget contenitore, La finestra.

6.1. gtk.Window()

Nel capitolo precedente abbiamo già visto questo widget in azione, aggiungiamo ora dei nuovi contenuti sviluppando la nostra prima finestra.

La gerarchia

```
+-- GObject
  +-- gtk.Object
    +-- GtkWidget
      +-- gtk.Container
        +-- gtk.Bin
          +-- gtk.Window
```

Il costruttore

```
gtk.Window(type=gtk.WINDOW_TOPLEVEL)
```

type: Il tipo di window

il parametro type può assumere i seguenti valori:

gtk.WINDOW_TOPLEVEL: Una window che non ha parents ed è solitamente corredata da un frame e da una top bar per l'iterazione con il window manager, le window toplevel sono le applicazioni principali e le finestre di dialogo.

gtk.WINDOW_POPUP: Una finestra che è ignorata dal window manager, solitamente utilizzata per tooltip e menu

6.1.1. Associare e leggere un titolo dalla window

Per associare un titolo alla finestra creata esiste un apposito metodo, `set_title()`.

```
gtk.Window.set_title(title)
```

title: Titolo da associare alla window, apparirà sulla top bar.

Il metodo `set_title()` assegna alla proprietà `title` della `gtk.Window()` il valore passato a `title`. Il titolo di ogni finestra apparirà nella top bar. Su X window il titolo sarà applicato dal window manager, quindi non si può sapere a priori come quest'ultimo potrà apparire all'utente, dipende molto dalla configurazione della macchina, il titolo scelto dovrebbe essere appropriato e diverso per ogni finestra del programma.

Per farlo apparire nella nostra finestra basterà quindi modificare il listato precedente aggiungendo una semplice linea di codice

```
self.win.set_title('Primo programma!')
```



Al contrario per leggere il titolo associato alla window bastera' utilizzare il metodo seguente:

```
gtk.Window.get_title(title)
```

Il metodo `get_title()` restituira' quindi il valore della proprieta' `title` associato alla finestra.

6.1.2. Decidere la posizione della window sullo schermo

```
gtk.Window.set_position(position)
```

position: La posizione che dovra' assumere la window

il parametro **position** puo' assumere i seguenti valori:

gtk.WIN_POS_NONE: Nessuna influenza sul posizionamento.

gtk.WIN_POS_CENTER: La window sara' piazzata al centro dello schermo.

gtk.IN_POS_MOUSE: La windows sara' piazzata in corrispondenza della attuale posizione del mouse.

gtk.WIN_POS_CENTER_ALWAYS: Mantiene la window centrata anche in caso di ridimensionamento ecc...

gtk.IN_POS_CENTER_ON_PARENT: Centra la window sul suo parent transitorio.

6.1.3. I widget transitori

I metodi che seguono gestiscono la transitoriet  della Window. Il concetto di transitorio puo' essere leggermente difficile all'inizio anche o solamente) per il nome che hanno scelto per noi... Un widget transitorio rimarra' sempre al di sopra di un altro widget, anche cliccando su quello sottostante il transitorio rimarra' in rilievo. Il primo metodo applica la transitoriet  ad una finestra, questa rimarra' quindi sempre in rilievo.

```
gtk.Window.set_transient_for(setting)
```

Passando come valore **None** sara' rimossa un'eventuale transitoriet .

Il secondo metodo invece interroga la finestra, ritornando il parent oppure None se la finestra non e' transitoria.

```
gtk.Window.get_transient_for()
```

6.1.4. Distruggere la finestra con i widget transitori

```
gtk.Window.set_destroy_with_parent(setting)
```

setting: se TRUE distrugge la window con i suoi parent transitori.

Il metodo `set_destroy_with_parent()` assegna alla proprieta' `destroy-with-parent` il valore passato a `setting`. Se TRUE, distrugge i parent transitori della window ed inoltre la window stessa. Questo metodo e' utile per distruggere i dialog che non devono persistere alla chiusura della window principale a cui essi sono associati.

6.1.5. Gestire il ridimensionamento della finestra

```
gtk.Window.set_resizable(resizable)
```

Il metodo `set_resizable()` assegna alla proprieta' `resizable` il valore passato a `resizable`. Se TRUE, lo user potra' ridimensionare la window, per default le window sono create ridimensionabili.

6.1.6. Muovere la finestra

```
gtk.Window.set_gravity(gravity)
```

gravity: Il valore gravity della window

il parametro **gravity** puo' assumere i seguenti valori:

gtk.gdk.GRAVITY_NORTH_WEST

gtk.gdk.GRAVITY_NORTH

gtk.gdk.GRAVITY_NORTH_EAST

gtk.gdk.GRAVITY_WEST

gtk.gdk.GRAVITY_CENTER

gtk.gdk.GRAVITY_EAST

gtk.gdk.GRAVITY_SOUTH_WEST

gtk.gdk.GRAVITY_SOUTH

gtk.gdk.GRAVITY_SOUTH_EAST

gtk.gdk.GRAVITY_STATIC

Il metodo `set_gravity()` assegna alla proprieta' `gravity` il valore passato. Il default e' **gtk.gdk.GRAVITY_NORTH_WEST** ovvero l'angolo in alto a sinistra, che in parole povere significa fai quello che vuoi...

E' bene precisare che la posizione indicata non sara' quella della finestra in relazione allo schermo, bensì e' il punto di partenza delle coordinate che possono essere passate da altri metodi come vedremo piu' avanti, in pratica il punto 0.0 di un ipotetico piano cartesiano.

Questo metodo non e' molto utile se utilizzato da solo, il metodo sara' utile in combinazione con **move(x, y)**

PyGTK GUI programming

```
gtk.Window.move(x, y)
```

Applicando questo metodo, la finestra si sposterà del valore delle coordinate passate ai parametri **x** e **y**.

E' comunque da tenere presente il fatto che il window manager può disinteressarsi di questo posizionamento.

Riprendendo il codice della finestra di esempio e modificandolo opportunamente vedremo il risultato.

Example 6.1. Spostare una finestra con le coordinate

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

class FirstWin:
    def __init__(self):
        self.win = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.win.set_gravity(gtk.gdk.GRAVITY_CENTER)
        self.win.move(700, 700)

        self.win.show()

    def main(self):
        gtk.main()

if __name__ == "__main__":
    first = FirstWin()
    first.main()
```

window_move.py (../pyhtml/window_move.html)

Con questo metodo fissiamo il punto di origine delle coordinate al centro della finestra

Passiamo le coordinate al metodo `move()`

Per capire appieno l'effetto che hanno questi metodi sul posizionamento della finestra, commentate inizialmente il metodo `move()` e controllate il posizionamento della finestra sullo schermo, in seguito applicate più volte il metodo cambiando ogni volta il valore delle coordinate

Potremmo avere la necessità di conoscere la posizione della finestra, magari per poterla spostare in seguito ad un evento quale può essere la pressione di un bottone, oppure per riportarla nella medesima posizione in caso fosse stata spostata dall'utente, per ottenere quindi le coordinate rispetto al suo punto 0,0 (quello settato da `gravity`) possiamo utilizzare il metodo `get_position()`. Questo metodo restituisce una tupla con le coordinate della finestra.

Anche per questo metodo come per altri visti in precedenza e' impossibile stabilire a priori se il risultato ottenuto e' valido al 100%, il window manager potrebbe trarre in inganno PyGTK, il quale farà di tutto per restituire il risultato più credibile.

La direzione del piano cartesiano sarà dall'alto verso il basso e da sinistra verso destra

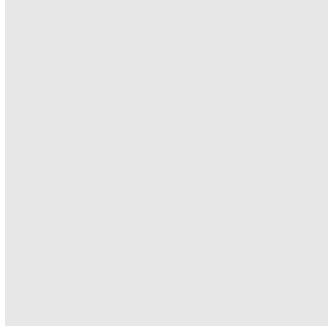
6.1.7. Settare le decorazioni della finestra

```
gtk.Window.set_decorated(setting)
```

setting: Se TRUE, applica la decorazione alla finestra.

Il metodo `set_decorated()` assegna alla flag `decorated` il valore passato a `setting`. Se `TRUE`, la window sara' corredata dalla top bar con i bottoni per il ridimensionamento, la chiusura, ecc... Per default le window sono corredate dala top bar. Se `FALSE` PyGTK fara' il possibile per convincere il window manager a creare una finestra senza bordi.

Il seguente e' un esempio di `set_decorated(gtk.FALSE)`, non molto utile vero..?



6.1.8. Settare le dimensioni della finestra

```
gtk.Window.set_default_size(width, height)
```

width: La larghezza in pixel, oppure `-1` per disabilitare il default.

height: L' altezza in pixel, oppure `-1` per disabilitare il default.

Il metodo `set_default_size()` assegna la larghezza e l'altezza di default ai valori passati a `width` e `height`. Se le dimensioni naturali della finestra (quelle richieste dai widget contenuti) sono piu' grandi del default, quest' ultimo sara' ignorato.

I metodi elencati sono solo alcuni tra quelli disponibili per il widget `gtk.Window()`, ma sicuramente sono quelli fondamentali per definire la geometria di una finestra.

6.1.9. Inserire un' icona nelle decorazioni della Window

E' possibile inserire un'icona nella finestra creata, la posizione che potra' assumere quest'ultima dipende molto dal window manager che si sta utilizzando, potrebbe posizionarsi nel frame della finestra oppure essere utilizzata nella barra quando la finestra viene minimizzata.

Prendiamo il codice di esempio della prima finestra e modifichiamo la classe `FirstWin()` come segue

```
class FirstWin:
    def __init__(self):
        self.win = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.icon = self.win.render_icon(gtk.STOCK_DIALOG_INFO, gtk.ICON_SIZE_BUTTON)
        self.win.set_icon(self.icon)
        self.win.show()
```

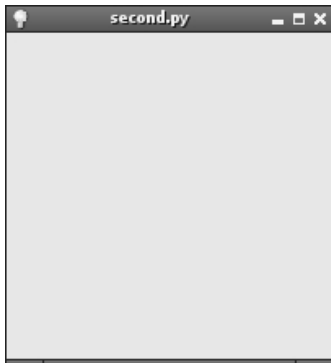
Come dicevamo precedentemente l'effetto dipende dal window manager, nel mio caso (fluxbox) l'icona apparira' solo nella minimizzazione della finestra



lo stesso programma su windows fara' apparire l'icona anche nel frame della finestra



Anche su un altro window manager come openbox l'icona comparira' nel frame della finestra



Quest'ultimo metodo non vale solo per il widget `gtk.Window()` ma e' applicabile a tutti i widget che discendono dalla classe padre `widget`, vedremo piu' in dettaglio questi metodi in un apposito capitolo.

6.1.10. Segnali ed eventi della Window

Anche ad un widget come `Window` possono essere connessi segnali ed eventi, bisogna ammettere che alcuni di questi difficilmente saranno mai utilizzati dalle nostre applicazioni, quelli che hanno qualche chance di essere utilizzati sono elencati qui sotto, gli altri sono sempre e comunque reperibili sulle API.

"activate-default"

Questo segnale e' emesso quando un widget all'interno della `Window` viene attivato alla pressione dei tasti `Enter` o `Return`

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

class SecondWin:
    def __init__(self):
        self.win = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.win.connect("delete_event", self.delete_event)
        self.win.connect("destroy", self.destroy)
        self.win.connect("activate-default", self.doFocus)
        self.vbox = gtk.VBox()
        self.win.add(self.vbox)
        self.button = gtk.Button("Focus to me!")
        self.vbox.pack_start(self.button)
        self.button2 = gtk.Button("Focus to me!")
        self.vbox.pack_start(self.button2)
        self.win.show_all()

    def doFocus(self, widget, data=None):
```

PyGTK GUI programming

```
print "widget -> ", widget

def delete_event(self, widget, event, data=None):
    return gtk.FALSE

def destroy(self, widget, data=None):
    return gtk.main_quit()

def main(self):
    gtk.main()

if __name__ == "__main__":
    second = SecondWin()
    second.main()
```

win_activate_default.py (../pyhtml/win_activate_default.html)

Solo alla pressione dei tasti Enter e Return il segnale sara' emesso dalla window, cio' nonostante l'attivazione dei Button reagira' anche al click del mouse o della barra spaziatrice.

"activate-focus"

Questo segnale e' emesso quando un widget all'interno della Window viene attivato alla pressione della barra spaziatrice.

```
self.win.connect("activate-focus", self.doFocus)
```

win_activate_focus.py (../pyhtml/win_activate_focus.html)

"move-focus"

Questo segnale e' emesso quando vengono premuti: i tasti direzionali, il tasto tab e la combinazione di tasti shift+tab.

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

class FirstWin:
    def __init__(self):
        self.win = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.win.connect("move-focus", self.doFrame)
        self.win.connect("destroy", lambda w: gtk.main_quit())
        self.win.show_all()

    def doFrame(self, widget, direction, data=None):
        print "direction -> ", direction

    def main(self):
        gtk.main()

if __name__ == "__main__":
    first = FirstWin()
    first.main()
```

win_activate_focus.py (../pyhtml/win_move_focus.html)

Questo segnale invia la direzione corrispondente al tasto premuto, da notare quindi il paramentro aggiuntivo alla funzione di callback.

"set-focus"

Questo segnale e' emesso quando il focus cambia a favore di un widget all'interno della Window.

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

class SecondWin:
    def __init__(self):
        self.win = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.win.connect("delete_event", self.delete_event)
        self.win.connect("destroy", self.destroy)
        self.win.connect("set-focus", self.doFocus)
        self.win.set_resizable(gtk.FALSE)
        self.win.set_title('wow un bottone!')
        self.win.set_border_width(5)
        self.vbox = gtk.VBox()
        self.win.add(self.vbox)
        self.button = gtk.Button("Focus to me!")
        self.vbox.pack_start(self.button)
        self.entry = gtk.Entry()
        self.entry.set_text("Focus to me!")
        self.vbox.pack_start(self.entry)
        self.button2 = gtk.Button("Focus to me!")
        self.vbox.pack_start(self.button2)
        self.win.show_all()

    def doFocus(self, window, widget, data=None):
        print "widget -> ", widget

    def delete_event(self, widget, event, data=None):
        return gtk.FALSE

    def destroy(self, widget, data=None):
        return gtk.main_quit()

    def main(self):
        gtk.main()

if __name__ == "__main__":
    second = SecondWin()
    second.main()
```

win_activate_focus.py (../pyhtml/win_set_focus.html)

6.1.11. Le proprieta' della Window

Quelle che seguono sono le proprieta' del widget Window, ricordo che a questo widget possono essere applicate anche quelle dei widget da cui discende, come da gerachia.

Table 6.1. Le proprieta'

Proprieta'	Lettura	Scrittura	Descrizione	Default	Introd
"accept-focus"	Si	Si	Se TRUE la finestra puo' ricevere il focus	TRUE	GTK+
"allow-grow"	Si	Si		TRUE	GTK+

PyGTK GUI programming

			Se TRUE la finestra puo' essere ingrandita oltre la dimensione minima.		
"allow-shrink"	Si	Si	Se TRUE la finestra non avra' dimensioni minime (pessima idea).	FALSE	GTK+
"decorated"	Si	Si	Se TRUE la finestra sara' decorata dal window manager.	TRUE	GTK+
"default-height"	Si	Si	L'altezza di default utilizzata quando la finestra e' mostrata sul display.	-1	GTK+
"default-width"	Si	Si	L'altezza di default utilizzata quando la finestra e' mostrata sul display.	-1	GTK+
"destroy-with-parent"	Si	Si	Se TRUE la finestra sara' distrutta quando lo saranno i suoi parent.	FALSE	GTK+
"focus-on-map"	Si	Si	Se TRUE la finestra potra' ricevere il focus quando mappata.	TRUE	GTK+
"gravity"	Si	Si	Il valore di gravity della finestra.	gtk.gdk.GRAVITY_NORTH_WEST	GTK+
"has-toplevel-focus"	Si	No	Se TRUE la finestra il focus e' all'interno della finestra.	FALSE	GTK+
"icon"	Si	Si		--	GTK+

PyGTK GUI programming

			L'icona per la finestra.		
"icon-name"	Si	Si	Il nome dell'icona per questa finestra, dipende dal tema utilizzato.	---	GTK+
"is-active"	Si	No	Se TRUE la finestra e' il widget attivo.	FALSE	GTK+
"modal"	Si	Si	Se TRUE le altre finestre non sono utilizzabili.	FALSE	GTK+
"resizable"	Si	Si	Se TRUE la finestra puo' essere ridimensionata.	TRUE	GTK+
"role"	Si	Si	Identificativo da usare quando si recupera una sessione.	None	GTK+
"screen"	Si	Si	Lo screen dove questa finestra sara' mostrata.	---	GTK+
"skip-pager-hint"	Si	Si	Se TRUE la finestra non comparira' nel pager.	FALSE	GTK+
"skip-taskbar-hint"	Si	Si	Se TRUE la finestra non comparira' nella task bar.	FALSE	GTK+
"title"	Si	Si	Il titolo della finestra che comparira' nel frame.	None	GTK+
"type"	Si	Si	Il tipo di finestra.	WINDOW_TOPLEVEL	GTK+
"type-hint"	Si	Si	Valore utile al WM per capire il tipo di finestra desiderata.	gtk.gdk.WINDOW_TYPE_HINT_NORMAL	GTK+
"window-position"	Si	Si	La posizione iniziale della finestra.	gtk.WIN_POS_NONE	GTK+

6.1.12. Gli attributi della Window

Quelli che seguono sono gli attributi del widget Window, ricordo che ad esso possono essere applicati anche quelli dei widget da cui discende come possiamo vedere dalla sua gerarchia.

Table 6.2. Gli attributi

Attributo	Letture	Scrittura	Descrizione
"allow_grow"	Si	No	Se TRUE la finestra puo' essere ingrandita oltre la dimensione minima.
"allow_shrink"	Si	No	Se TRUE la finestra non ha dimensione minima.
"configure_notify_received"	Si	No	Se TRUE la finestra ha ricevuto un evento di ridimensionamento.
"configure_request_count"	Si	No	Il numero di configurazioni richieste.
"decorated"	Si	No	Se TRUE la finestra ha le decorazioni applicate dal WM.
"default_widget"	Si	No	Il widget figlio che sara' attivato per default.
"focus_widget"	Si	No	Il widget figlio che ha il focus.
"frame"	Si	No	Il frame (se presente) della gtk.gdk.Window
"frame_bottom"	Si	No	L'altezza della parte bassa del frame.
"frame_left"	Si	No	La larghezza della parte sinistra del frame.
"frame_right"	Si	No	La larghezza della parte destra del frame.
"frame_top"	Si	No	L'altezza della parte alta del frame.
"gravity"	Si	No	Il valore di gravity della finestra.
"group"	Si	No	Il gruppo (se esiste) al quale la finestra appartiene.
"has_focus"	Si	No	Se TRUE la finestra ha il focus.
"has_frame"	Si	No	Se TRUE la finestra ha il frame.
"has_user_ref_count"	Si	No	Se TRUE la finestra non e' stata distrutta.
"iconify_initially"	Si	No	Se TRUE la finestra e' stata ridotta ad icona.
"keys_changed_handler"	Si	No	L>ID dell'idle handler utilizzato per i cambi nei gruppi degli acceleratori.
"maximize_initially"	Si	No	Se TRUE la finestra e' stata massimizzata.
"mnemonic_modifier"	Si	No	Il mnemonic modify utilizzato tramite un tasto per attivare l'accelerator.
"modal"	Si	No	Se TRUE la finestra e' modale.
"need_default_position"	Si	No	-----
"need_default_size"	Si	No	-----
"position"	Si	No	La posizione iniziale della finestra
"stick_initially"	Si	No	Se TRUE la finestra e' stata resa sticky.

PyGTK GUI programming

"title"	Si	No	Il titolo della finestra.
"transient_parent"	Si	No	Il parent transitorio della finestra.
"type"	Si	No	Il tipo di finestra (gtk.WINDOW_TOPLEVEL o gtk.WINDOW_POPUP).
"type_hint"	Si	No	Il tipo di hint della finestra
"wmclass_class"	Si	No	Il tipo di window system class hint della finestra.
"wmclass_name"	Si	No	Il window system name hint della finestra.
"wm_role"	Si	No	L'identificativo univoco della finestra.

Chapter 7. I bottoni

Il widget `Button` non e' l'unico bottone utilizzabile nelle GUI scritte con PyGTK, ma e' il widget padre di altri tipi di bottone, come possiamo vedere dalla loro gerarchia.

7.1. La gerarchia dei bottoni

```
+-- gobject.GObject
+-- gtk.Object
+-- gtk.Widget
+-- gtk.Container
+-- gtk.Bin
+-- gtk.Button
+-- gtk.ColorButton
+-- gtk.FontButton
+-- gtk.ToggleButton
+-- gtk.CheckButton
+-- gtk.RadioButton
```

`gtk.ColorButton` e `gtk.FontButton` sono disponibili solo dalla versione 2.4 e superiore di PyGTK, alla pressione del primo verra' aperto il widget `gtk.ColorSelectionDialog`, il secondo apre `gtk.Font.SelectionDialog`. Questi ultimi vengono utilizzati in particolar modo quando devono essere settate delle preferenze, piu' avanti analizzeremo in dettaglio questi widget.

7.2. `gtk.Button()`

Dopo aver discusso del funzionamento di alcuni segnali e della costruzione del widget `Window` andiamo ad applicare le nozioni fin qui apprese, implementando con un bottone la nostra precedente applicazione. Creeremo inoltre una funzione di callback e applicheremo dei segnali.

Innanzitutto vediamo come implementare la nostra precedente applicazione:

Example 7.1. Primo esempio di `Button`

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

class SecondWin:
    def __init__(self):
        self.win = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.win.connect("delete_event", self.delete_event)
        self.win.connect("destroy", self.destroy)
        self.win.set_resizable(gtk.FALSE)
        self.win.set_title('wow un bottone!')
        self.win.set_border_width(5)
        self.win.show()
        self.button = gtk.Button("Hello World")
        self.button.connect("clicked", self.hello, None)
        self.button.connect_object("clicked", gtk.Widget.destroy, self.win)
        self.win.add(self.button)
        self.button.show()

    def hello(self, widget, data=None):
        print "Hello, World"

    def delete_event(self, widget, event, data=None):
        return gtk.FALSE
```

```
def destroy(self, widget, data=None):
    return gtk.main_quit()

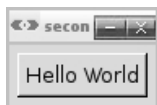
def main(self):
    gtk.main()

if __name__ == "__main__":
    second = SecondWin()
    second.main()
```

first_button.py (../pyhtml/first_button.html)

Lanciando questo secondo script, comparira' una nuova finestra con un bottone simile a questa:

Figure 7.1. La finestra contenente le proprieta' dei widget



Il nuovo codice presenta una serie di novita', innanzitutto abbiamo settato due degli innumerevoli attributi di Window, piu' precisamente l'abbiamo resa immutabile nelle dimensioni con `set_resizable()` ed abbiamo applicato un bordo, il quale non e' altro che lo spazio tra il bottone e il contorno della finestra, questo metodo e' spiegato piu' in dettaglio nel capitolo dedicato ai metodi comuni, ad ogni modo il suo funzionamento come vedete e' di facile comprensione.

Procedendo nel listato abbiamo creato le funzioni callback di distruzione del widget, ovvero **`delete_event()`** e **`destroy()`**, abbiamo inoltre creato una funzione che alla pressione del bottone stampera' sullo stdout il messaggio 'Hello World'.

Le funzioni di uscita dal programma sono state applicate sia alla finestra che al widget, ora chiudendo l'applicazione, sia alla pressione del bottone sia utilizzando l'apposito tasto dal window manager, anche la funzione `gtk.mainloop()` verra' interrotta.

Le funzioni **`delete_event()`** e **`destroy()`** potrebbero sembrare ridondanti ma in realta' non lo sono affatto, `delete_event()` come potete vedere nell'esempio soprastante, restituisce l'operatore booleano `FALSE` quest'ultimo chiederà al window manager di chiudere la finestra, se quest'ultimo fosse invece settato a `TRUE` la finestra non avrebbe alcuna reazione rimanendo aperta in attesa di ulteriori istruzioni. L'opzione `TRUE` quindi potrà essere associata ad un evento che faccia aprire un messaggio popup, per chiedere ad esempio all'utente la conferma di chiusura, in caso di risposta affermativa potremo tramite la funzione `destroy()`, far chiudere la finestra e di conseguenza uscire da `gtk.mainloop()`.

Come già precedentemente descritto parlando degli eventi, il segnale emesso dal widget alla pressione del bottone, verra' catturato dalla classe padre `gobject.GObject` ed eseguita la funzione ad esso associata, in questo caso `destroy()`.

Il costruttore

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

label: Il testo che dovrà comparire all'interno del bottone

stock: Il testo e l'immagine che dovrà comparire all'interno del bottone

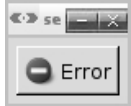
use_underline: La lettera che sarà utilizzata come mnemonic accelerator

7.2.1. Utilizzare gli Stock Item

Nel programma precedente l'unico attributo utilizzato nella creazione del bottone e' la Label, al secondo attributo del costruttore, ovvero stock, possiamo passare il nome contenuto in un set di immagini e testo preconfezionati, questi ultimi possono essere inseriti nei Button utilizzando questa forma del costruttore:

```
gtk.Button(None, gtk.STOCK_DIALOG_ERROR)
```

e questo e' il risultato:



7.2.2. Utilizzare i mnemonic accelerator

Il metodo **use_underline** serve per assegnare un mnemonic accelerator alla Label presente nel Button. Il suo utilizzo e' molto semplice, basta inserire un underscore nella label in corrispondenza di una qualsiasi lettera, quest'ultimo sara' accessibile da tastiera con la combinazione alt+lettera, inoltre andra' settato il parametro **use_underline** a TRUE. Il parametro **use_underline** ha fatto da poco la sua comparsa, e' stato recentemente inserito nel toolkit ovvero dalla versione 2.4, eccone un esempio pratico:

```
gtk.Button('_Hello World', gtk.TRUE)
```



Se volessimo inserire un underscore nel testo della label, bisognera' semplicemente utilizzare un doppio underscore:

```
self.button = gtk.Button("Hello__World")
```

oppure e' possibile utilizzare la funzione `set_use_underline()`

```
self.button = gtk.Button("Hello World")
self.button.set_use_underline(gtk.FALSE)
```



7.2.3. I segnali emessi da Button

I bottoni emettono dei segnali quando l'utente interagisce con essi, nella stesura del codice questi segnali per essere utili devono venire catturati e indirizzati a delle funzioni di callback. Questi segnali sono 5 e per la precisione vengono emessi quando:

enter: il puntatore entra nel bottone

leave: il puntatore esce dal bottone

clicked: il bottone viene cliccato

pressed: il bottone viene premuto

released: il bottone viene rilasciato

PyGTK GUI programming

Nonostante sia possibile utilizzare tutti questi segnali, l'unico usato dalla maggioranza delle applicazioni e' clicked che abbiamo gia' visto all'opera nell'esempio precedente, piu' precisamente nelle linee di codice qui riportate:

```
self.button.connect("clicked", self.hello, None)
self.button.connect_object("clicked", gtk.Widget.destroy, self.win)
```

Il segnale emesso dal bottone sara' catturato e la funzione o le funzioni di callback ad esso associate saranno eseguite, nel nostro caso self.hello() e gtk.Widget.destroy().

I segnali emessi dai bottoni sono anche dei metodi degli stessi, in quanto tali possono essere eseguiti all'interno di altre funzioni, di seguito un ulteriore esempio:

Example 7.2. Emettere un segnale

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

class SecondWin:
    def __init__(self):
        self.win = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.win.connect("delete_event", self.delete_event)
        self.win.connect("destroy", self.destroy)
        self.win.set_resizable(gtk.FALSE)
        self.win.set_title('wow un bottone!')
        self.win.set_border_width(5)
        self.vbox = gtk.VBox()
        self.win.add(self.vbox)
        self.button = gtk.Button("click me")
        self.button.set_focus_on_click(gtk.FALSE)
        self.button.connect("clicked", self.click, None)
        self.vbox.pack_start(self.button)
        self.button1 = gtk.Button("I will be clicked")
        self.button1.connect("clicked", self.hello, None)
        self.button1.connect_object("clicked", gtk.Widget.destroy, self.win)
        self.vbox.pack_start(self.button1)
        self.win.show_all()

    def hello(self, widget, data=None):
        print "Hello, World"

    def click(self, widget, data=None):
        self.button1.clicked()

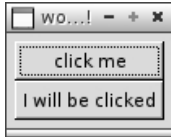
    def delete_event(self, widget, event, data=None):
        return gtk.FALSE

    def destroy(self, widget, data=None):
        return gtk.main_quit()

    def main(self):
        gtk.main()

if __name__ == "__main__":
    second = SecondWin()
    second.main()
```

button_connect.py (../pyhtml/button_connect.html)



Tralasciamo per il momento questo metodo di impacchettare i widget.

Facciamo eseguire al bottone, quando cliccato, la funzione `click()`.

Creiamo il nuovo bottone che subira' il click.

Facciamo eseguire al nuovo bottone, quando cliccato, la funzione `hello()`.

Facciamo eseguire al bottone, quando cliccato, la funzione `destroy()`.

Provate ad eseguire il programma, indipendentemente dal bottone che sara' cliccato l'applicazione si comportera' allo stesso modo. Ovviamente un utilizzo di questo tipo e' inutile, serve solo a capire come emettere un segnale.

7.2.4. Le proprieta' del Button

Quelle che seguono sono le proprieta' del widget Button, ricordo che a questo widget possono essere applicate anche quelle dei widget da cui discende, come da gerachia.

Table 7.1. Le proprieta'

Proprieta'	Lettura	Scrittura	Descrizione	Default	Introdotta
"focus-on-click"	Si	Si	Se TRUE il bottone riceve il focus quando cliccato.	TRUE	GTK+ 2.4
"label"	Si	Si	Il testo della Label contenuta nel bottone.	--	GTK+ 2.0
"relief"	Si	Si	L'effetto di rilievo applicato al bottone.	gtk.RELIEF_NORMAL	GTK+ 2.0
"use-underline"	Si	Si	Se TRUE la Label all'interno del bottone avra' i mnemonic accelerator attivi.	TRUE	GTK+ 2.0
"use-stock"	Si	Si	Se TRUE il testo sara' usato per prendere uno stock invece di essere mostrato come label.	TRUE	GTK+ 2.0
"xalign"	Si	Si	Se il child del bottone e' un gtk.Misc o gtk.Alignment questo valore sara' usato come allineamento orizzontale.	--	GTK+ 2.4
"yalign"	Si	Si	Se il child del bottone e' un gtk.Misc o gtk.Alignment questo valore sara' usato come allineamento verticale.	--	GTK+ 2.4

7.3. gtk.ToggleButton()

Questi 3 bottoni derivano da Button sono infatti a lui molto simili, la differenza fondamentale e' quella di poter assumere due stati, normale ed attivo, hanno inoltre caratteristiche comuni, ToggleButton e' la base su cui sono costruiti, molte delle sue caratteristiche infatti valgono anche per i restanti due.

La gerarchia

```
+-- GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.Bin
          +-- gtk.Button
            +-- gtk.ToggleButton
```

Il costruttore

```
gtk.ToggleButton(label=None, use_underline=TRUE)
```

Osservando il costruttore una prima differenza evidente da Button consiste nel non poter inserire gtk.Stock, gli attributi label e use_underline invece saranno utilizzati allo stesso modo.

7.3.1. Un primo esempio di ToggleButton

Un' ulteriore differenza consiste nella gestione dei segnali, quello emesso da ToggleButton prende il nome di toggled, per meglio chiarire tutti questi nuovi concetti possiamo ad un esempio pratico, il nuovo programma avra' al suo interno tre bottoni, due ToggleButton ed un semplice Button che esegue la funzione destroy().

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

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

class SecondWin:
    def __init__(self):

        self.win = gtk.Window(gtk.WINDOW_TOPLEVEL)

        self.win.connect("delete_event", self.delete_event)
        self.win.connect("destroy", self.destroy)
        self.win.set_resizable(gtk.FALSE)
        self.win.set_border_width(10)
        self.win.show()

        self.vbox = gtk.VBox(gtk.TRUE, 3)
        self.win.add(self.vbox)
        self.vbox.show()

        self.button_t1 = gtk.ToggleButton("primo toggle")
        self.button_t1.connect("toggled", self.tog, "primo toggle")
        self.button_t2 = gtk.ToggleButton("secondo toggle")
        self.button_t2.connect("toggled", self.tog, "secondo toggle")
        self.button = gtk.Button(None, gtk.STOCK_QUIT)
        self.button.connect("clicked", self.destroy)

        self.vbox.pack_start(self.button_t1, gtk.TRUE, gtk.TRUE, 5)
        self.vbox.pack_start(self.button_t2, gtk.TRUE, gtk.TRUE, 5)
        self.vbox.pack_start(self.button, gtk.TRUE, gtk.TRUE, 5)
```

PyGTK GUI programming

```
self.button_t1.show()
self.button_t2.show()
self.button.show()

def tog(self, widget, data=None):
    print "%s e' ora %s" % (data, ("OFF", "ON")[widget.get_active()])

def delete_event(self, widget, event, data=None):
    return gtk.FALSE

def destroy(self, widget, data=None):
    return gtk.main_quit()

def main(self):
    gtk.main()

if __name__ == "__main__":
    second = SecondWin()
    second.main()
```

button_toggle.py (../pyhtml/button_toggle.html)



Provate a cliccare uno dei due toggle button, il segnale verra' catturato e reindirizzato alla funzione di callback tog(), sul *term verra' quindi stampata la stringa che riporta lo status del widget, infine per uscire da gtk.mainloop() bastera' cliccare sul bottone quit.

7.3.2. I segnali di ToggleButton

ToggleButton discendendo gerarchicamente da Button ne eredita anche i segnali, l'unico segnale aggiunto e' il **"toggled"** che abbiamo gia' visto in azione nell'esempio precedente.

Ad ogni modo non ha molto senso connettere un segnale come "clicked" ad un ToggleButton

7.3.3. Le proprieta' del ToggleButton

Quelle che seguono sono le proprieta' del widget ToggleButton, ricordo che a questo widget possono essere applicate anche quelle dei widget da cui discende, come da gerachia.

Table 7.2. Le proprieta'

Proprieta'	Letture	Scrittura	Descrizione	Default	Introdotta
"active"	Si	Si	Se TRUE il bottone dovrebbe essere premuto.	FALSE	GTK+ 2.0
"inconsistent"	Si	Si	Se TRUE il bottone si trova in uno stato "intermedio".	FALSE	GTK+ 2.0

"draw-indicator"	Si	Si	Se TRUE la parte toggle del bottone e' in display.	FALSE	GTK+ 2.0
------------------	----	----	--	-------	----------

7.3.4. Gli attributi del ToggleButton

Quelli che seguono sono gli attributi del widget ToggleButton, ricordo che ad esso possono essere applicati anche quelli dei widget da cui discende come possiamo vedere dalla sua gerarchia.

Table 7.3. Gli attributi

Attributo	Letture	Scrittura	Descrizione
"draw_indicator"	Si	No	Se TRUE la parte toggle del bottone e' in display.

7.4. gtk.CheckButton()

Il secondo bottone ovvero CheckButton e' concettualmente molto simile al precedente.

La gerarchia

```
+-- gobject.GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.Bin
          +-- gtk.Button
            +-- gtk.ToggleButton
              +-- gtk.CheckButton
```

Il costruttore

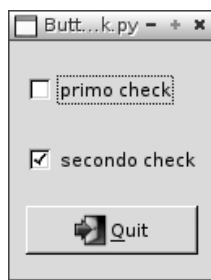
```
gtk.CheckButton(label=None, use_underline=TRUE)
```

Il costruttore e' identico a quello del ToggleButton.

7.4.1. Un primo esempio di CheckButton

I metodi sono identici a quelli del ToggleButton, come si puo' notare aprendo il link sottostante.

[button_check.py \(../pyhtml/button_check.html\)](#)



Provate a cliccare uno dei due CheckButton, il segnale verra' catturato e reindirizzato alla funzione di callback tog(), sul *term verra' quindi stampata la stringa che riporta lo status del widget, infine per uscire da gtk.mainloop() bastera' cliccare sul bottone quit.

7.4.2. Segnali, proprieta' ed attributi del CheckButton

CheckButton eredita gli stessi segnali, proprieta' ed attributi dal ToggleButton.

7.5. gtk.RadioButton()

Il terzo bottone ovvero RadioButton non ha grandi differenze rispetto ai precedenti se utilizzato singolarmente, se invece vengono inseriti piu' RadioButton si formera' un gruppo. Il gruppo puo' comodamente essere usato per far scegliere allo user una o piu' opzioni all'interno della GUI

La gerarchia

```
+-- GObject
+-- gtk.Object
+-- gtk.Widget
+-- gtk.Container
+-- gtk.Bin
+-- gtk.Button
+-- gtk.ToggleButton
+-- gtk.CheckButton
+-- gtk.RadioButton
```

Il costruttore

```
gtk.RadioButton(group=None, label=None, use_underline=TRUE)
```

group: Un altro RadioButton che e' il riferimento al gruppo, oppure None se e' il primo ad essere creato.

label: Il testo che deve comparire vicino al RadioButton.

use_underline: Come abbiamo gia' visto in precedenza, se TRUE assegnera' un mnemonic accelerator al RadioButton

7.5.1. Un primo esempio di RadioButton

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

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

class SecondWin:
    def __init__(self):

        self.win = gtk.Window(gtk.WINDOW_TOPLEVEL)

        self.win.connect("delete_event", self.delete_event)
        self.win.connect("destroy", self.destroy)
        self.win.set_resizable(gtk.FALSE)
        self.win.set_border_width(10)

        self.vbox = gtk.VBox(gtk.TRUE, 3)
        self.win.add(self.vbox)
        self.vbox.show()

        self.button_r1 = gtk.RadioButton(None, "primo radio", gtk.FALSE)
        self.button_r1.connect("toggled", self.tog, "primo radio")
        self.button_r2 = gtk.RadioButton(self.button_r1, "secondo radio")
        self.button_r2.connect("toggled", self.tog, "secondo radio")
        self.button_r3 = gtk.RadioButton(self.button_r1, "terzo radio")
```

PyGTK GUI programming

```
self.button_r3.connect("toggled", self.tog, "terzo radio")
self.button = gtk.Button(None, gtk.STOCK_QUIT)
self.button.connect("clicked", self.destroy)

self.vbox.pack_start(self.button_r1, gtk.TRUE, gtk.TRUE, 5)
self.vbox.pack_start(self.button_r2, gtk.TRUE, gtk.TRUE, 5)
self.vbox.pack_start(self.button_r3, gtk.TRUE, gtk.TRUE, 5)
self.vbox.pack_start(self.button, gtk.TRUE, gtk.TRUE, 5)

self.win.show_all()

def tog(self, widget, data=None):
    print "%s e' ora %s" % (data, ("OFF", "ON")[widget.get_active()])

def delete_event(self, widget, event, data=None):
    return gtk.FALSE

def destroy(self, widget, data=None):
    return gtk.main_quit()

def main(self):
    gtk.main()

if __name__ == "__main__":
    second = SecondWin()
    second.main()
```

button_radio_multi.py (./pyhtml/button_radio_multi.html)



Cliccando sui RadioButton si avra' l'effetto desiderato di attivazione/disattivazione, nel *term saranno stampate le informazioni sullo stato dei bottoni.

7.5.2. Leggere e cambiare i gruppi di appartenenza.

I RadioButton possono essere senza alcun problema spostati da un gruppo di bottoni ad un altro, per capire a quale gruppo attualmente appartengono si utilizza il seguente metodo:

```
gtk.RadioButton.get_group()
```

Sara' restituita una lista di RadioButton presenti all'interno dello stesso gruppo, per spostare un RadioButton in un altro gruppo invece si utilizza un altro metodo ovvero:

```
gtk.RadioButton.set_group(group)
```

group: Un altro RadioButton facente parte del gruppo dove il nostro bottone dovra' essere inserito.

7.5.3. I segnali di RadioButton

RadioButton discendendo gerarchicamente da Button ne eredita anche i segnali.

"group-changed"

Il segnale **group-changed** viene emesso quando il RadioButton viene spostato da un gruppo di appartenenza ad un altro, tramite i metodi descritti precedentemente.

Anche per il RadioButton non ha molto senso connettere un segnale come "clicked".

7.5.4. Le proprietà del RadioButton

Quelle che seguono sono le proprietà del widget RadioButton, ricordo che a questo widget possono essere applicate anche quelle dei widget da cui discende, come da gerachia.

Table 7.4. Le proprietà

Proprietà'	Lettura	Scrittura	Descrizione	Default	Introdotta
"group"	No	Si	Il gruppo di appartenenza del RadioButton.	--	GTK+ 2.0

7.5.5. Gli attributi del ToggleButton

RadioButton non ha attributi propri ma eredita quelli dei widget che lo precedono nella gerachia.

Chapter 8. Metodi, attributi, proprieta' e segnali delle classi widget, container e bin

Le classi widget, container e bin contengono metodi, attributi, segnali e proprieta' che vengono molto spesso utilizzate dai widget che da queste discendono gerarchicamente, per questa ragione e' molto importante imparare, capire e ricordare le gerarchie.

8.1. La classe widget

La classe widget gerarchicamente e' forse la piu' importante, da essa infatti discendono tutti i widget intesi come oggetti con cui interagire, sono quindi esclusi oggetti come TextBuffer e Tooltips, discendono infatti da gobject_GObject il primo e da gtk.Object il secondo.

I metodi di questa classe sono moltissimi e coprono una vasta gamma di utilizzi, in particolare:

- I metodi per il drag and drop
- I metodi per la gestione degli eventi
- I metodi per mappare, realizzare e mostrare i widget
- I metodi per il dimensionamento
- I metodi per modificare gli stili
- I metodi per accedere alle risorse

I metodi che riguardano drag and drop e stili verranno trattati piu' avanti in quanto richiedono piu' dimestichezza con il toolkit.

8.1.1. Impadronirsi del controllo

I metodi che seguono si impadroniscono e rilasciano il controllo, impedendo l'iterazione da parte dell'utente con altri oggetti all'interno dello stesso contenitore, come ad esempio all'interno di una Window.

Nel prossimo esempio questo metodo sara' applicato ad un Button.

Example 8.1.

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

class SecondWin:
    def __init__(self):
        self.win = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.win.connect("delete_event", self.delete_event)
        self.win.connect("destroy", self.destroy)
        self.win.set_resizable(gtk.FALSE)
        self.win.set_title('wow un bottone!')
        self.win.set_border_width(5)
        self.vbox = gtk.VBox()
        self.win.add(self.vbox)
        self.button = gtk.Button("No Focus")
        self.vbox.pack_start(self.button)
        self.button1 = gtk.Button("Focus")
        self.button1.grab_add()
        self.vbox.pack_start(self.button1)
        self.button2 = gtk.Button("No Focus")
        self.vbox.pack_start(self.button2)
```

PyGTK GUI programming

```
self.win.show_all()

def delete_event(self, widget, event, data=None):
    return gtk.FALSE

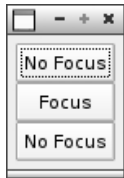
def destroy(self, widget, data=None):
    return gtk.main_quit()

def main(self):
    gtk.main()

if __name__ == "__main__":
    second = SecondWin()
    second.main()
```

grab_add.py (../pyhtml/grab_add.html)

Eseguendo il programma comparira' una finestra simile alla seguente, sara' impossibile cliccare altri Button se non quello centrale che si e' impadronito totalmente del controllo.



Per fare in modo che l'oggetto rilasci il controllo bastera' utilizzare successivamente il metodo **grab_remove()**, la forma standard e' la seguente.

```
gtk.Widget.grab_remove()
```

Ovviamente ogni volta che incontriamo la forma standard del metodo trattato, per applicarla ai nostri programmi dovremmo sostituire `gtk.Widget` con il nome dell' oggetto che stiamo utilizzando, nel caso precedente bisogna utilizzare:

```
self.button1.grab_remove()
```

Chapter 9. Scendiamo di un livello, il wrapper GDK

La libreria GDK si pone tra il server X e la libreria GTK+ risparmiando il lavoro ingrato di comunicare con le Xlib, un soggetto non molto facile.

Questa libreria contiene gli oggetti base che ci permettono di comunicare in maniera trasparente con il window manager, ad esempio una Window spesso contiene una o piu' gtk.gdk.Window, il motivo e' presto spiegato. La maggior parte dei widget infatti e' dotata di una propria gtk.gdk.Window, inserendo ad esempio i widget in una Window di tipo Toplevel, otterremo il risultato di avere molte gtk.gdk.Window all'interno di una Window.

Ma vediamo ora gli oggetti fondamentali della libreria GDK

9.1. gtk.gdk.Window()

Come detto in precedenza questa e' la base su cui sono costruiti la maggior parte dei widget, in pratica si tratta di una regione rettangolare dello schermo che puo' essere mostrata o nascosta, nella libreria Xlib queste funzioni vengono chiamate mapping e unmapping.

Queste regioni possono catturare degli eventi, essere mosse e ridimensionate e sono organizzate in una struttura gerarchica, piu' gtk.gdk.Window possono essere contenute all'interno di un'altra gtk.gdk.Window e cosi' via.

Le Window sono profondamente diverse dalle gtk.gdk.Window, le prime infatti solitamente non hanno un genitore ma sono fatte per contenere altri oggetti. Quasi tutti i widget come dicevamo prima hanno al loro interno una gtk.gdk.Window, esistono delle eccezioni come ad esempio la Label questi ultimi per poter ricevere degli eventi hanno bisogno di un widget provvisto di una gtk.gdk.Window che li contenga ovvero EventBox

La gerarchia

```
+-- GObject
  +-- gtk.gdk.Drawable
    +-- gtk.gdk.Window
```

Per il momento non guardiamo il loro costruttore, e' molto improbabile durante lo sviluppo di una GUI dover creare delle gtk.gdk.Window, molto probabilmente vorremo solamente raccogliere il nome della gtk.gdk.Window associata ad un widget per poterla manipolare.

9.1.1. Catturare una gtk.gdk.Window

Per far questo possiamo utilizzare l'attributo window dei widget, sara' molto semplice catturare la nostra gtk.gdk.Window

Vediamo di capire meglio di che cosa si tratta con un esempio pratico:

Example 9.1. Catturare una gtk.gdk.Window

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

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

PyGTK GUI programming

```
class FirstWin(gtk.Window):
    def __init__(self):
        gtk.Window.__init__(self, gtk.WINDOW_TOPLEVEL)
        self.connect("destroy", lambda w: gtk.main_quit())
        self.show()
        gdkwin = self.window
        print gdkwin

    def main(self):
        gtk.main()

if __name__ == "__main__":
    first = FirstWin()
    first.main()
```

gdk_window_get.py (../pyhtml/gdk_window_get.html)

assegnamo alla variabile gdkwin l'oggetto gtk.gdk.Window tramite l'attributo window

Stampiamo l'output in un *term

Eseguendo il programma verra' quindi stampato nel *term l'oggetto gtk.gdk.Window, qualcosa di simile a:

```
<gtk.gdk.Window object (GdkWindow) at 0xb7ccac0c>
```

9.2. gtk.gdk.Cursor()

Ogni gtk.gdk.Window puo' essere dotata di un gtk.gdk.Cursor questo non e' altro che un immagine bitmap che indica la posizione del mouse, per default il cursore utilizzato e' quello della parent Window

Il costruttore

```
gtk.gdk.Cursor(cursor_type)
```

cursor_type: Il tipo di cursore da utilizzare

9.2.1. Cambiare il cursore di una gtk.gdk.Window

Cambiare il cursore ad un widget e' relativamente semplice, basta catturare la sua gtk.gdk.Window e cambiarle il cursore.

Example 9.2. Cambiare il cursore ad un widget

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

class FirstWin(gtk.Window):
    def __init__(self):
        gtk.Window.__init__(self, gtk.WINDOW_TOPLEVEL)
        self.show()
        gdkwin = self.window
        cursor = gtk.gdk.Cursor(gtk.gdk.SPIDER)
        gdkwin.set_cursor(cursor)

    def main(self):
        gtk.main()

if __name__ == "__main__":
```

```
first = FirstWin()
first.main()
```

gdk_window_change_cursor.py (../pyhtml/gdk_window_change_cursor.html)

Costruiamo un nuovo oggetto `gtk.gdk.Cursor` con un simpatico....ragno!
Applichiamo il nuovo cursore alla `gtk.gdk.Window` che abbiamo catturato precedentemente.

9.3. `gtk.gdk.Keymap()`

`gtk.gdk.Keymap` e' lo strumento principale con cui catturare la pressione dei tasti, traducendo il segnale che arriva dalla tastiera ad un valore interpretabile.

Nonostante ci possano essere piu' `gtk.gdk.Keymap` ovvero una per ogni display, PyGTK supporta un solo display e di conseguenza una sola `gtk.gdk.Keymap`.

La gerarchia

```
+-- GObject.GObject
+-- gtk.gdk.Keymap
```

`gtk.gdk.Keymap` non ha un suo costruttore, per catturare la `gtk.gdk.Keymap` devono essere utilizzati i metodi degli esempi che seguono.

9.3.1. Catturare una `gtk.gdk.Keymap`

Per catturare l'oggetto `gtk.gdk.Keymap` utilizziamo il metodo seguente:

Example 9.3. Catturare una `gtk.gdk.Keymap`

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

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

class FirstWin(gtk.Window):
    def __init__(self):
        gtk.Window.__init__(self, gtk.WINDOW_TOPLEVEL)
        self.connect("destroy", lambda w: gtk.main_quit())
        self.show()
        gdkkey = gtk.gdk.keymap_get_default()
        print gdkkey

    def main(self):
        gtk.main()

if __name__ == "__main__":
    first = FirstWin()
    first.main()
```

gdk_keymap_get.py (../pyhtml/gdk_keymap_get.html)

assegnamo alla variabile `gdkkey` l'oggetto `gtk.gdk.Keymap` tramite il metodo **`gtk.gdk.keymap_get_default()`**
Stampiamo l'output in un `*term`

PyGTK GUI programming

Eseguendo il programma verra' quindi stampato nel *term l'oggetto `gtk.gdk.Keymap`, qualcosa di simile a:

```
<gtk.gdk.KeymapX11 object (GdkKeymapX11) at 0xb7ccaaf4>
```

Chapter 10. Un ulteriore widget fondamentale, la Label

10.1. gtk.Label()

Un altro dei widget fondamentali utilizzati nelle GUI sono le Label, utilizzerò sempre il termine inglese in quanto la traduzione etichette sinceramente non mi pare appropriata :)

Le Label fanno parte della categoria di widget che non hanno una propria X-window, da non confondersi con la Window, sono denominati comunemente windowless, il che significa che non possono rispondere a degli eventi; Per ovviare a questo problema, si può utilizzare uno speciale contenitore, quale ad esempio il widget `gtk.EventBox()`.

Come già precedentemente visto per Button il testo di una Label può contenere dei mnemonic accelerator, il metodo non è applicabile solamente nel caso della label contenuta in un bottone.

Di seguito un piccolo esempio di Label che amplieremo strada facendo.

Example 10.1. Primo esempio di Label

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

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

class SecondWin:
    def __init__(self):
        self.win = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.win.connect("delete_event", self.delete_event)
        self.win.connect("destroy", self.destroy)
        self.label = gtk.Label("Hello (cruel) World...")
        self.win.add(self.label)
        self.win.show_all()

    def delete_event(self, widget, event, data=None):
        return gtk.FALSE

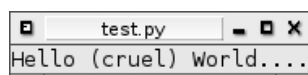
    def destroy(self, widget, data=None):
        return gtk.main_quit()

    def main(self):
        gtk.main()

if __name__ == "__main__":
    second = SecondWin()
    second.main()
```

first_label.py (../pyhtml/first_label.html)

Eseguito il programma comparirà una finestra simile alla seguente



La gerarchia

```

+-- GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Misc
        +-- gtk.Label

```

Il costruttore

```
gtk.Label(text=None)
```

text: Il testo che deve comparire nella label

10.1.1. Inserire il testo nella Label

```
gtk.Label.set_text(text)
```

text: il testo che sara' inserito nella Label

Questo metodo e' utile nel caso si voglia cambiare una Label precedentemente impostata con un altro valore, ad esempio:

Example 10.2. Inserire o cambiare il testo nella Label

```

#!/usr/bin/env python

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

class SecondWin:
    def __init__(self):
        self.win = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.win.connect("delete_event", self.delete_event)
        self.win.connect("destroy", self.destroy)
        self.vbox = gtk.VBox(gtk.TRUE, 0)
        self.win.add(self.vbox)
        self.label = gtk.Label("Hello (cruel) World...")
        self.button = gtk.Button(None, gtk.STOCK_EXECUTE)
        self.button.connect("clicked", self.change_text)
        self.vbox.pack_start(self.label)
        self.vbox.pack_start(self.button)
        self.win.show_all()

    def change_text(self, widget, data=None):
        self.label.set_text("Hello (better) World...")

    def delete_event(self, widget, event, data=None):
        return gtk.FALSE

    def destroy(self, widget, data=None):
        return gtk.main_quit()

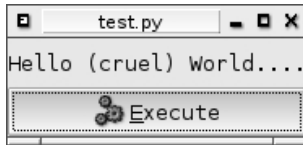
    def main(self):
        gtk.main()

if __name__ == "__main__":
    second = SecondWin()
    second.main()

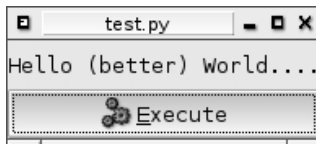
```

label_change_text.py (../pyhtml/label_change_text.html)

Questo programma quando eseguito si presentera' con la label costruita all'avvio



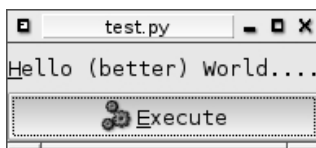
e dopo aver cliccato il bottone la label sara' cambiata prendendo il nuovo valore



Bisogna tener presente che il nuovo testo inserito, annullera' tutti i mnemonic accelerator precedentemente inseriti, per inserire un nuovo mnemonic accelerator, bisognera' modificare la funzione change_text() del precedente listato come segue:

```
def change_text(self, widget, data=None):
    self.label.set_text("_Hello (better) World...")
    self.label.set_use_underline(gtk.TRUE)
```

oppure in alternativa potra' essere utilizzato un altro metodo, ovvero `gtk.Label.set_text_with_mnemonic()` come vedremo tra poco.



Attenzione che se il mnemonic accelerator non e' associato a nessuna funzione, un `GtkWarning` sara' restituito:

```
GtkWarning: Couldn't find a target for a mnemonic activation. gtk.main()
```

10.1.2. Utilizzare i mnemonic accelerator

```
gtk.Label.set_text_with_mnemonic(text)
```

text: Il testo comprensivo di underscore che sara' inserito nella Label

Questa funzione e' utile quando si voglia associare un mnemonic accelerator ad un widget esterno alla label stessa, in questo caso va richiamato un ulteriore metodo, il prossimo che vedremo, il quale associa il mnemonic accelerator ad un ulteriore widget.

10.1.3. Associare un mnemonic accelerator ad un widget esterno

```
gtk.Label_set_mnemonic_widget(widget)
```

widget: e' l'oggetto che deve essere associato con il mnemonic accelerator.

Un esempio puo' essere il seguente, il mnemonic accelerator fa attivare il checkbox a cui la label fa riferimento

Example 10.3.

```

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

class SecondWin:
    def __init__(self):
        self.win = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.win.set_border_width(10)
        self.win.connect("delete_event", self.delete_event)
        self.win.connect("destroy", self.destroy)
        self.hbox = gtk.HBox(gtk.TRUE, 10)
        self.win.add(self.hbox)
        self.label = gtk.Label("_Switch me on")
        self.label.set_use_underline(gtk.TRUE)
        self.button = gtk.CheckButton(None, None)
        self.label.set_mnemonic_widget(self.button)
        self.button.connect("toggled", self.switch)
        self.hbox.pack_start(self.label)
        self.hbox.pack_start(self.button)
        self.win.show_all()

    def switch(self, widget, data=None):
        (data, ("OFF", "ON")[widget.get_active()])

    def delete_event(self, widget, event, data=None):
        return gtk.FALSE

    def destroy(self, widget, data=None):
        return gtk.main_quit()

    def main(self):
        gtk.main()

if __name__ == "__main__":
    second = SecondWin()
    second.main()

```

label_set_mnemonic.py (../pyhtml/label_set_mnemonic.html)

Alla pressione dei tasti alt+s il checkbutton sara' attivato/disattivato.

**10.1.4. Leggere il testo associato ad una Label**

```
gtk.Label.get_text()
```

Questo metodo e' in realta' molto semplice, restituisce il valore di una Label

10.1.5. Utilizzare il Pango markup in una Label

Per evidenziare il contenuto di una Label e' possibile utilizzare Pango markup, fondamentalmente sono tre i metodi disponibili per utilizzarlo, nell'ordine:

```

gtk.Label.set_markup(text)
gtk.Label.set_use_markup(mode)
gtk.Label.set_markup_with_mnemonic(text)

```

Passa il testo contenente il markup alla Label

Se TRUE permette ad una Label già' inizializzata di utilizzare Pango markup al suo interno.

Passa il testo contenente il markup alla Label, sarà inoltre possibile utilizzare i mnemonic accelerator.

Example 10.4.

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

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

class SecondWin:
    def __init__(self):
        self.win = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.win.set_border_width(10)
        self.win.connect("delete_event", self.delete_event)
        self.win.connect("destroy", self.destroy)
        self.label = gtk.Label("<b><big>I am big and bold...</big></b>")
        self.label.set_use_markup(gtk.TRUE)
        self.win.add(self.label)
        self.win.show_all()

    def delete_event(self, widget, event, data=None):
        return gtk.FALSE

    def destroy(self, widget, data=None):
        return gtk.main_quit()

    def main(self):
        gtk.main()

if __name__ == "__main__":
    second = SecondWin()
    second.main()
```

label_set_markup.py (../pyhtml/label_set_markup.html)

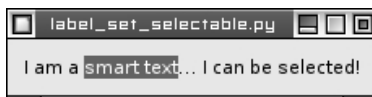


10.1.6. Operazioni di selezione in una Label

Per default il testo nelle Label non è selezionabile, per renderlo tale bisogna utilizzare il metodo seguente:

```
gtk.Label.set_selectable(gtk.TRUE)
```

label_set_selectable.py (../pyhtml/label_set_selectable.html)



Oltre alla selezione del testo fatta dallo user, il testo di una Label può anche essere selezionato come segue:

```
gtk.Label.select_region(start, end)
```

start, end: il numero di carattere di inizio e fine della selezione.

Questo metodo non puo' essere eseguito in fase di creazione della Label ma quest'ultima deve gia' essere in display. Diversamente un `GtkWarning` sara' restituito e il testo non sara' inserito nella clipboard.

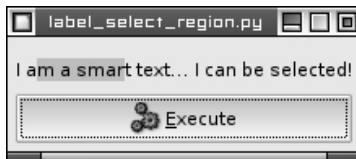
```
GtkWarning: file gtkwidget.c: line 7344 (gtk_widget_get_clipboard): assertion `gtk_widget_has_clipboard` failed: 0
GtkWarning: file gtkclipboard.c: line 561 (gtk_clipboard_set_with_owner): assertion `clipboard` failed: 0
```

Quello che possiamo fare quindi e' selezionare il testo tramite una funzione chiamata successivamente:

Example 10.5. Selezionare il testo

```
def selectRegion(self, widget):
    self.label.select_region(3, 11)
```

label_select_region.py (../pyhtml/label_select_region.html)



Se ora provate ad incollare dalla clipboard in un editor o in altra applicazione, la regione selezionata sara' restituita.

Impostando il valore `-1` alla selezione, questa andra' fino alla fine della Label

E' anche possibile ottenere il valore del range dei caratteri selezionati dall'utente con il seguente metodo:

```
gtk.Label.get_selection_bounds()
```

Il valore restituito e' una tupla contenete i valori **start**, **end**

Modifichiamo in questo modo la classe usata precedentemente

```
class SecondWin:
    def __init__(self):
        self.win = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.win.set_border_width(5)
        self.win.connect("delete_event", self.delete_event)
        self.win.connect("destroy", self.destroy)
        self.vbox = gtk.VBox(gtk.TRUE, 0)
        self.win.add(self.vbox)
        self.label = gtk.Label("I am a smart text... I can be selected!")
        self.label.set_selectable(gtk.TRUE)
        self.button = gtk.Button(None, gtk.STOCK_EXECUTE)
        self.button.connect("clicked", self.lookRegion)
        self.labelValues = gtk.Label()
        self.vbox.pack_start(self.label)
        self.vbox.pack_start(self.button)
        self.vbox.pack_start(self.labelValues)
        self.win.show_all()
```

Associamo al bottone la funzione di callback che leggerà i valori della selezione

Label in cui stamperemo i valori della selezione

E creiamo una nuova funzione di callback che faccia le due operazioni richieste:

```
def lookRegion(self, widget):
    self.data = self.label.get_selection_bounds()
```

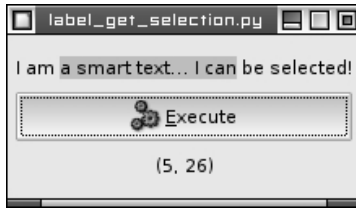
PyGTK GUI programming

```
self.labelValues.set_text(str(self.data))
```

leggiamo i valori della selezione associandoli a `self.data`.

stampiamo i valori ottenuti nella label fittizia, convertendoli preventivamente in una stringa.

`label_get_selection.py` (`../pyhtml/label_get_selection.html`)



Chapter 11. Interagire con l'utente

11.1. gtk.Entry()

Interagire con l'utente e' sicuramente un compito fondamentale per una GUI, le Entry sono lo strumento principale per svolgere questo compito.

Le Entry a differenza delle Label fanno parte dei widget che possiedono una propria `gtk.gdk.window`, ecco un esempio di Entry che sara' ampliato strada facendo.

Example 11.1. Primo esempio di Entry

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

class SecondWin:
    def __init__(self):
        self.win = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.win.connect("delete_event", self.delete_event)
        self.win.connect("destroy", self.destroy)
        self.entry = gtk.Entry(30)
        self.win.add(self.entry)
        self.win.show_all()

    def delete_event(self, widget, event, data=None):
        return gtk.FALSE

    def destroy(self, widget, data=None):
        return gtk.main_quit()

    def main(self):
        gtk.main()

if __name__ == "__main__":
    second = SecondWin()
    second.main()
```

first_entry.py (../pyhtml/first_entry.html)

Eseguendo il programma comparira' una finestra con all'interno una Entry come la seguente:



La gerarchia

```
+-- GObject
+-- gtk.Object
+-- gtk.Widget
++-- gtk.Entry
```

Il costruttore

```
gtk.Entry(MaxLetters=0)
```

MaxLetters: Il numero massimo di caratteri che saranno accettati

Il range del numero di caratteri varia da 0 a 65536, settare il valore a 0 significa non assegnare un valore massimo

Il numero massimo di caratteri puo' essere settato anche successivamente con il metodo `set_max_lenght()`

```
gtk.Entry.set_max_lenght(MaxLetters)
```

Il valore di `MaxLetters` puo' essere rintracciato tramite il metodo `get_max_lenght()`

```
gtk.Entry.get_max_lenght()
```

11.1.1. Leggere e scrivere il testo

Quello che segue e' forse da considerarsi il metodo fondamentale di questo widget, ovvero **`get_text`**

```
gtk.Entry.get_text(text)
```

text: e' il testo che e' stato inserito dall'utente nella Entry

Passiamo subito ad un esempio pratico:

Example 11.2. Prendere l'input e assegnarlo ad una variabile

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

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

class SecondWin:
    def __init__(self):
        self.win = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.win.connect("delete_event", self.delete_event)
        self.win.connect("destroy", self.destroy)
        self.entry = gtk.Entry(30)
        self.entry.connect("activate", self.getUserInput)
        self.win.add(self.entry)
        self.win.show_all()

    def getUserInput (self, widget):
        userinput = self.entry.get_text()
        print userinput

    def delete_event(self, widget, event, data=None):
        return gtk.FALSE

    def destroy(self, widget, data=None):
        return gtk.main_quit()

    def main(self):
        gtk.main()

if __name__ == "__main__":
    second = SecondWin()
    second.main()
```

entry_get_text.py (../pyhtml/entry_get_text.html)

Il costruttore della Entry, il valore massimo di caratteri e' settato a 30.

Il metodo connect lancia la funzione di callback quando attivato, in questo caso alla pressione del tasto enter.

La dichiarazione della funzione di callback.

Il cuore del programma, prende il testo inserito nella Entry e lo assegna alla variabile userinput.

Funzione fittizia solo per verificare che il contenuto sia stato davvero assegnato.

11.1.2. Inserire il testo nella Entry

E' possibile inserire nella Entry un testo di default sia al momento della creazione che successivamente. Il metodo per poter eseguire questa operazione e' il seguente:

```
gtk.Entry.set_text(text)
```

text: e' il testo che sara' inserito nella Entry

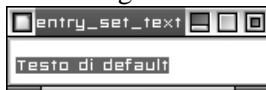
In questo esempio il testo sara' inserito in fase di creazione della Entry

Example 11.3. Inserire il testo nella Entry

```
class SecondWin:
    def __init__(self):
        self.win = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.win.connect("delete_event", self.delete_event)
        self.win.connect("destroy", self.destroy)
        self.entry = gtk.Entry(30)
        self.entry.set_text("Testo di default")
        self.win.add(self.entry)
        self.win.show_all()
```

entry_set_text.py (../pyhtml/entry_set_text.html)

Con questo metodo il testo sara' inserito come default alla creazione del widget



E' inoltre possibile con una funzione di callback, cambiare il testo di default successivamente. Per farlo creiamo un bottone che lanci la funzione di callback:

Example 11.4. Cambiare il testo nella Entry

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

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

class SecondWin:
    def __init__(self):
        self.win = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.win.connect("delete_event", self.delete_event)
        self.win.connect("destroy", self.destroy)
        self.vbox = gtk.VBox()
```

PyGTK GUI programming

```
self.entry = gtk.Entry(30)
self.entry.set_text("default")
self.vbox.pack_start(self.entry, gtk.TRUE, gtk.TRUE, 0)
self.button = gtk.Button(None, gtk.STOCK_EXECUTE)
self.button.connect("clicked", self.changeText)
self.vbox.pack_start(self.button, gtk.TRUE, gtk.TRUE, 0)
self.win.add(self.vbox)
self.win.show_all()

def changeText(self, widget):
    self.entry.set_text("Nuovo testo!")

def delete_event(self, widget, event, data=None):
    return gtk.FALSE

def destroy(self, widget, data=None):
    return gtk.main_quit()

def main(self):
    gtk.main()

if __name__ == "__main__":
    second = SecondWin()
    second.main()
```

entry_change_text.py (./pyhtml/entry_change_text.html)

La funzione di callback sara' eseguita alla pressione del bottone

La funzione di callback che cambiera' il testo nella Entry

Alla pressione del tasto la funzione di callback sara' chiamata e il testo nella Entry cambiato in questo modo:



11.1.3. Rendere invisibile il testo

Durante la fase di iterazione con lo user puo' capitare di dover immettere una password, in questo caso il testo digitato non dovra' essere in chiaro ma rimpiazzato da asterischi, il metodo per poterlo fare e' il seguente:

```
gtk.Entry.set_visibility(mode)
```

mode: settando il valore di mode a FALSE il testo sara' rimpiazzato da asterischi

Example 11.5. Rendere invisibile il testo

```
class SecondWin:
    def __init__(self):
        self.win = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.win.connect("delete_event", self.delete_event)
        self.win.connect("destroy", self.destroy)
        self.entry = gtk.Entry(30)
        self.entry.set_visibility(gtk.FALSE)
        self.entry.connect("activate", self.getUserInput)
        self.win.add(self.entry)
        self.win.show_all()
```

entry_set_invisible.py (../pyhtml/entry_set_invisible.html)



Purtroppo non posso dirvi cosa si nasconde dietro quegli asterischi... :)

Il metodo per verificare se la Entry e' impostata a visible oppure no, e' il seguente:

```
gtk.Entry.get_visibility()
```

Questo metodo restituisce lo stato del widget, se il valore restituito e' gtk.TRUE il testo e' visibile.

Nel caso non vi piacciono gli asterichi, il carattere con cui il testo sara' reso invisibile puo' essere cambiato. Per farlo si utilizza il metodo seguente:

```
gtk.Entry.set_invisible_char(char)
```

char: il carattere con cui verranno sostituiti gli asterischi

Supponiamo di voler utilizzare la chiocciola invece dell'asterisco, presto fatto...

Example 11.6. Cambiare il carattere di default

```
class SecondWin:
    def __init__(self):
        self.win = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.win.connect("delete_event", self.delete_event)
        self.win.connect("destroy", self.destroy)
        self.entry = gtk.Entry(30)
        self.entry.set_visibility(gtk.FALSE)
        self.entry.set_invisible_char('@')
        self.entry.connect("activate", self.getUserInput)
        self.win.add(self.entry)
        self.win.show_all()
```

set_invisible_char.py (../pyhtml/set_invisible_char.html)

Il carattere '@' sostituirà l'asterisco durante la digitazione.



11.1.4. Impostare il numero massimo di caratteri visualizzabili

Un altro valore che possiamo impostare e' il numero massimo di caratteri visualizzabili all'interno della Entry, il metodo e' il seguente:

```
gtk.Entry.set_width_chars(chars)
```

chars: il numero massimo di caratteri visualizzabili

Example 11.7. Numero di caratteri in display

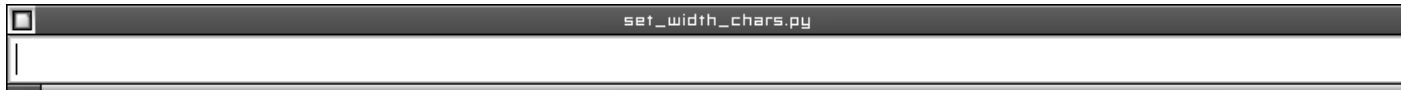
```
class SecondWin:
```

PyGTK GUI programming

```
def __init__(self):
    self.win = gtk.Window(gtk.WINDOW_TOPLEVEL)
    self.win.connect("delete_event", self.delete_event)
    self.win.connect("destroy", self.destroy)
    self.entry = gtk.Entry(0)
    self.entry.set_width_chars(100)
    self.win.add(self.entry)
    self.win.show_all()
```

set_width_chars.py (./pyhtml/set_width_chars.html)

Il risultato sara' il seguente:



Come il widget verra' visualizzato dipende anche dal metodo utilizzato per inserirlo nella GUI. Ovviamente e' anche possibile recuperare il valore impostato come numero massimo di caratteri e piu' precisamente con:

```
gtk.Entry.get_width_chars()
```

Il valore restituito sara' appunto il numero massimo di caratteri.

11.1.5. Modificare l'allineamento del testo

E' possibile, qualora il testo fosse piu' corto della Entry, modificarne l'allineamento in orizzontale

```
gtk.Entry.set_alignment(value)
```

value: numero in forma decimale indicante lo spostamento da sinistra verso destra.

Example 11.8.

```
class SecondWin:
    def __init__(self):
        self.win = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.win.connect("delete_event", self.delete_event)
        self.win.connect("destroy", self.destroy)
        self.entry = gtk.Entry(0)
        self.entry.set_width_chars(50)
        self.entry.set_alignment(0.5)
        self.win.add(self.entry)
        self.win.show_all()
```

entry_set_alignment.py (./pyhtml/entry_set_alignment.html)

Il risultato sara' il seguente:



Il valore deve essere una frazione di intero espresso in numero decimale, il valore 1 allineera' il testo verso destra.

Chapter 12. Messaggi di dialogo

12.1. gtk.Dialog()

Un altro modo per interagire con l'utente e' quello di far apparire dei messaggi di dialogo. Un esempio classico e' la finestra di pop up che chiede la conferma di chiudere l'applicazione.

Dialog e' un widget relativamente semplice. E' composto da 3 elementi "base", una window su cui sono stati inseriti un HBox ed un VBox in cui possiamo inserire altri widget.

Iniziamo con il vedere la gerarchia ed il costruttore, successivamente analizzeremo un esempio pratico.

La gerarchia

```
+-- GObject.GObject
+-- gtk.Object
   +-- gtk.Widget
       +-- gtk.Container
           +-- gtk.Bin
               +-- gtk.Window
                   +-- gtk.Dialog
```

Il costruttore

```
gtk.Dialog(title=None, parent=None, flags=0, (button_text_or_stock, response_or_integer))
```

title: Il titolo che comparira' nella decorazione della finestra del dialog

parent: Il widget padre, ovvero quello da cui e' nato il dialog

flags: Una o piu' flags tra: `gtk.DIALOG_MODAL`, `gtk.DIALOG_DESTROY_WITH_PARENT` o `gtk.DIALOG_NO_SEPARATOR`

(button_text_or_stock, response_or_integer): Una tupla che rappresenta bottone/responso, vedi piu' avanti per maggiori dettagli

12.1.1. Un primo esempio di Dialog

Come gia' menzionato all'inizio del capitolo una delle applicazioni pratiche per un Dialog e' chiedere la conferma allo user dell'uscita dal programma, ecco un pratico esempio:

Example 12.1. Sicuro di voler uscire?

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

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

class Dialog:
    def __init__(self):
        self.win = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.win.connect("delete_event", self.delete_event)
        self.win.connect("destroy", self.destroy)
        self.button = gtk.Button(None, gtk.STOCK_QUIT)
        self.button.connect("clicked", self.dialogRun)
```

PyGTK GUI programming

```
self.win.add(self.button)
self.win.show_all()

def dialogRun(self, widget):
    self.dialog = gtk.Dialog('Sure?', self.win,
                             gtk.DIALOG_MODAL | gtk.DIALOG_DESTROY_WITH_PARENT,
                             (gtk.STOCK_OK, gtk.RESPONSE_ACCEPT, gtk.STOCK_CANCEL, gtk.RESPONSE_REJECT))
    self.dialog.vbox.pack_start(gtk.Label("Vuoi davvero uscire??"))
    self.dialog.show_all()
    response = self.dialog.run()
    if response == gtk.RESPONSE_ACCEPT:
        return gtk.main_quit()
    elif response == gtk.RESPONSE_REJECT:
        self.dialog.destroy()
    self.dialog.destroy()

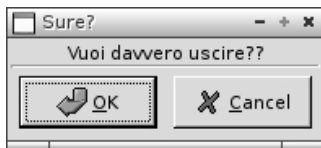
def delete_event(self, widget, event, data=None):
    return gtk.FALSE

def destroy(self, widget, data=None):
    return gtk.main_quit()

def main(self):
    gtk.main()

if __name__ == "__main__":
    dia = Dialog()
    dia.main()
```

dialog.py (../pyhtml/dialog.html)



Creiamo un bottone Quit fittizio, invece di chiamare direttamente la funzione destroy fara' aprire il Dialog.

Il metodo connect lancia la funzione di callback self.dialogRun.

Ecco finalmente il costruttore di Dialog.

Inseriamo una Label nel VBox gia' presente nel Dialog.

Mettiamo in display il Dialog e rimaniamo in attesa di una risposta da parte dell'utente.

Cliccando sul tasto OK la condizione response == gtk.RESPONSE_ACCEPT sara' true.

Di conseguenza sara' chiamato il gtk.main_quit() e l'applicazione sara' terminata.

Cliccando sul tasto CANCEL la condizione response == gtk.RESPONSE_REJECT sara' true.

Di conseguenza solo il Dialog sara' terminato.

Qualsiasi cosa succeda chiudi il Dialog ad esempio per la chiusura dal window manager.

12.2. gtk.MessageDialog()

MessageDialog come Dialog e' un widget relativamente semplice e potrebbe tranquillamente essere scritto utilizzando quest'ultimo, e' utile perche' in ogni caso permette di risparmiare un po' di... "digitazione"

Vediamo la gerarchia ed il costruttore, successivamente analizzeremo un esempio pratico.

La gerarchia

```
+++ gobject.GObject
```

12.2. gtk.MessageDialog()

```

+-- gtk.Object
  +-- gtk.Widget
    +-- gtk.Container
      +-- gtk.Bin
        +-- gtk.Window
          +-- gtk.Dialog
            +-- gtk.MessageDialog

```

Il costruttore

```
gtk.MessageDialog(parent=None, flags=0, type=gtk.MESSAGE_INFO, buttons=gtk.BUTTONS_NONE, mes
```

parent: Il widget padre, ovvero quello da cui e' nato il dialog oppure None

flags: Una o piu' flags tra: `gtk.DIALOG_MODAL` e `gtk.DIALOG_DESTROY_WITH_PARENT`

type: Il tipo di MessageDialog ovvero `gtk.MESSAGE_INFO`, `gtk.MESSAGE_WARNING`, `gtk.MESSAGE_QUESTION` oppure `gtk.MESSAGE_ERROR`.

buttons: Un set predefinito di bottoni tra `gtk.BUTTONS_NONE`, `gtk.BUTTONS_OK`, `gtk.BUTTONS_CLOSE`, `gtk.BUTTONS_CANCEL`, `gtk.BUTTONS_YES_NO`, `gtk.BUTTONS_OK_CANCEL`

message_format: Il testo che sara' inserito nel MessageDialog.

12.2.1. Un primo esempio di MessageDialog

Uno degli utilizzi piu' classici di MessageDialog e' quello di mostrare delle info allo user:

Example 12.2. Delle utili info

```

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

class MyDialog(gtk.Window):
    def __init__(self):
        gtk.Window.__init__(self, gtk.WINDOW_TOPLEVEL)
        self.connect("destroy", lambda w: gtk.main_quit())
        button = gtk.Button(None, gtk.STOCK_EXECUTE)
        button.connect("clicked", self.showMessage)
        self.add(button)
        self.show_all()

    def showMessage(self, widget, data=None):
        message = gtk.MessageDialog(None, gtk.DIALOG_MODAL, gtk.MESSAGE_INFO,
                                     gtk.BUTTONS_CLOSE, "This message is cool.. :)")
        message.show()
        resp = message.run()
        if resp == gtk.RESPONSE_CLOSE:
            message.destroy()

    def main(self):
        gtk.main()

if __name__ == "__main__":
    mess = MyDialog()
    mess.main()

```

message_dialog.py (../pyhtml/message_dialog.html)



Il costruttore in azione, e' molto intuitivo dopo avere visto il costruttore di Dialog.

gtk.RESPONSE_CLOSE e' il responso associato al bottone

gtk.BUTTONS_CLOSE.

12.2.2. Utilizzare il markup nella Label all'interno di un MessageDialog

Per poter utilizzare il pango markup language nella Label all'interno della MessageDialog bisogna utilizzare un po' di fantasia nel catturare il widget...vediamo come sempre un esempio pratico.

Example 12.3. Catturare la Label

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

class MyDialog(gtk.Window):
    def __init__(self):
        gtk.Window.__init__(self, gtk.WINDOW_TOPLEVEL)
        self.connect("destroy", lambda w: gtk.main_quit())
        button = gtk.Button(None, gtk.STOCK_EXECUTE)
        button.connect("clicked", self.showMessage)
        self.add(button)
        self.show_all()

    def showMessage(self, widget, data=None):
        message = gtk.MessageDialog(None, gtk.DIALOG_MODAL, gtk.MESSAGE_INFO,
                                    gtk.BUTTONS_CLOSE, "<b>This message is cool.. :)</b>")
        message.vbox.get_children()[0].get_children()[1].set_property("use-markup", True)
        message.show()
        resp = message.run()
        if resp == gtk.RESPONSE_CLOSE:
            message.destroy()

    def main(self):
        gtk.main()

if __name__ == "__main__":
    mess = MyDialog()
    mess.main()
```

message_dialog_markup.py (../pyhtml/message_dialog_markup.html)



PyGTK GUI programming

Scriviamo il messaggio utilizzando il pango markup language.

Catturiamo la Label all'interno del MessageDialog e applichiamo la proprietà "use-markup" a True.

Chapter 13. I tool widget e la statusbar

I widget che iniziano con tool non sono altro, come dice il nome stesso, che strumenti adatti per gli scopi piu' disparati. I rami principali sono 2 ovvero toolbar e tooltips, iniziamo con il vedere il primo.

13.1. gtk.Toolbar()

La Toolbar e' un widget utile nel caso si voglia sostituire o implementare un menu un maniera visuale. Gli oggetti nella Toolbar infatti sono di facile individuazione anche grazie all'applicazione di Tooltips che vedremo in seguito.

A partire dalla versione 2.4 di PyGTK l'interfaccia con la Toolbar e' cambiata in favore di un altro widget, ToolItem, anche questo sara' analizzato in dettaglio piu' avanti.

La gerarchia

```
+-- GObject.Object
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.Toolbar
```

Il costruttore

```
gtk.Toolbar()
```

13.1.1. Inserire un item nella Toolbar

```
gtk.Toolbar.insert(what, position)
```

what: un ToolItem che sara' inserito nella Toolbar

position: la posizione che dovra' assumere il widget inserito.

I valori che **position** potra' avere sono positivi o negativi, se positivi il conteggio partira' dall' inizio della Toolbar, se negativi ovviamente dalla fine.

L'esempio seguente mostra questo concetto, saranno inseriti due bottoni, per il costruttore di questi ultimi vi rimando alla lettura della sezione successiva.

Example 13.1. Inserire i bottoni nella Toolbar

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

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

class SecondWin:
    def __init__(self):
        self.win = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.win.connect("delete_event", self.delete_event)
        self.win.connect("destroy", self.destroy)
        self.win.set_default_size(250, 50)
        self.toolbar = gtk.Toolbar()
        self.toolButtQuit = gtk.ToolButton(gtk.STOCK_QUIT)
```

PyGTK GUI programming

```
self.toolButtQuit.connect("clicked", self.destroy)
self.toolbar.insert(self.toolButtQuit, -1)
self.toolButtNew = gtk.ToolButton(gtk.STOCK_NEW)
self.toolButtNew.connect("clicked", self.destroy)
self.toolbar.insert(self.toolButtNew, 0)
self.win.add(self.toolbar)
self.win.show_all()

def delete_event(self, widget, event, data=None):
    return gtk.FALSE

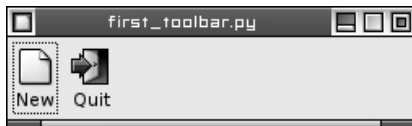
def destroy(self, widget, data=None):
    return gtk.main_quit()

def main(self):
    gtk.main()

if __name__ == "__main__":
    second = SecondWin()
    second.main()
```

first_toolbar.py (./pyhtml/first_toolbar.html)

Questo programma quando eseguito si presentera' con la label costruita all'avvio



13.2. gtk.Tooltips()

13.2. gtk.Tooltips()

I Tooltips sono molto utili nella descrizione di altri widget presenti all'interno delle applicazioni

La gerarchia

```
+-- GObject
  +-- gtk.Object
    +-- gtk.Tooltips
```

Il costruttore

```
gtk.Tooltips()
```

Il costruttore creera' un gruppo di Tooltips

13.2.1. Abilitare o disabilitare un gruppo di Tooltips

```
gtk.Tooltips.enable()
```

```
gtk.Tooltips.disable()
```

Questi due metodi come ovvio dal loro nome, abilitano e disabilitano un gruppo di Tooltips

13.2.2. Assegnare un Tooltips ad un widget;

`gtk.Tooltips.set_tip(widget, text, private=None)`

widget: Il widget a cui associare il Tooltips

text: Il testo che comparirà nella finestrella al passaggio del puntatore.

private: Un testo che non verrà messo in display ma che può essere usato in un context-sensitive help

L'esempio seguente applica un Tooltips ad un Button.

Example 13.2. Associare un Tooltips ad un bottone.

```
#!/usr/bin/env python
# -*- coding: utf8 -*-

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

class SecondWin:
    def __init__(self):
        self.win = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.win.connect("delete_event", self.delete_event)
        self.win.connect("destroy", self.destroy)
        self.win.set_resizable(gtk.FALSE)
        self.win.set_title('wow un bottone!')
        self.win.set_border_width(5)
        self.button = gtk.Button("Hello World")
        self.button.connect("clicked", self.hello, None)
        self.button.connect_object("clicked", gtk.Widget.destroy, self.win)
        self.tooltip = gtk.Tooltips()
        self.tooltip.set_tip(self.button, "questo è un bottone...", "tooltip assegnato al bottone")
        self.win.add(self.button)
        self.win.show_all()

    def hello(self, widget, data=None):
        print "Hello, World"

    def delete_event(self, widget, event, data=None):
        return gtk.FALSE

    def destroy(self, widget, data=None):
        return gtk.main_quit()

    def main(self):
        gtk.main()

if __name__ == "__main__":
    second = SecondWin()
    second.main()
```

first_tooltip.py (../pyhtml/first_tooltip.html)

Questo programma quando eseguito si presenterà con la label costruita all'avvio



13.3. gtk.Statusbar()

La Statusbar e' utilizzata solitamente per mettere in display informazioni utili, quali ad esempio il nome di un file ma anche il numero di righe e caratteri in un editor di testi, vedi ad esempio il Chapter 14, *Manipolare il testo*.

La gerarchia

```
+-- GObject.GObject
+-- gtk.Object
  +-- gtk.Widget
    +-- gtk.Container
      +-- gtk.Box
        +-- gtk.HBox
          +-- gtk.Statusbar
```

Il costruttore

```
gtk.Statusbar()
```

Chapter 14. Manipolare il testo

La manipolazione del testo e' un task spesso molto importante, PyGTK mette a disposizione la serie di widget che scopriremo strada facendo in questo capitolo.

Tutti i widget utilizzati per sviluppare una vero editor di testi sono strettamente legati tra di loro, quindi per dare un senso logico a questo capitolo, faro' una breve introduzione di tutti gli elementi che lo compongono per arrivare a mescolarli con una certa 'logigita', cosa che questa frase non ha ancora....

14.1. gtk.TextView()

TextView e' il widget che avra' al suo interno il testo da manipolare, a partire dalla versione 2.0 di GTK+ il testo e' codificato UTF-8. Questo non e' l'unico strumento che ci permettera' di manipolare i testi, via via scopriremo anche gli altri.

La gerarchia

```
+-- GObject
  +-- gtk.Object
    +-- gtk.Widget
      +-- gtk.Container
        +-- gtk.TextView
```

Il costruttore

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

buffer: Come vedremo piu' avanti il TextBuffer è il "contenitore" del nostro testo.

14.1.1. Un primo esempio di TextView

Costruire una GUI che possa contenere del testo e' relativamente semplice, basta costruire una nuova Window ed inserirvi all'interno un widget TextView, anche senza buffer.

Example 14.1. La creazione di una TextView

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

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

class MainWindow(gtk.Window):
    def __init__(self):
        gtk.Window.__init__(self)
        self.connect("delete_event", self.doQuit)
        self.set_default_size(300, 200)
        self.set_border_width(5)
        self.text = gtk.TextView()
        self.add(self.text)
        self.show_all()

    def doQuit(self, widget, data=None):
        gtk.main_quit()

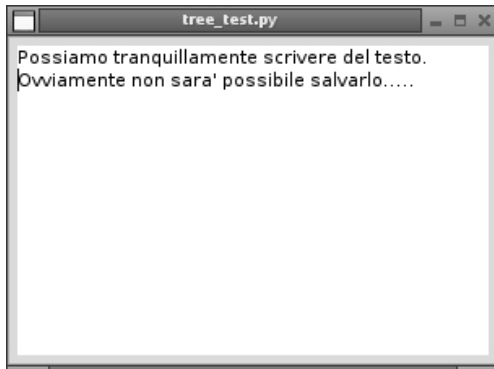
def main(self):
    gtk.main()
```

```
if __name__ == "__main__":
    go = MainWindow()
    go.main()
```

first_textview.py (../pyhtml/first_textview.html)

Per comodita' subclassiamo una Window, sara' possibile quindi utilizzare i suoi metodi come "self"
 Inizializziamo la Window per poterla utilizzare
 Creiamo una nuova TextView e' da notare la mancata assegnazione di un buffer.

Figure 14.1. La prima TextView



14.1.2. Associare e recuperare un TextBuffer dalla TextView

Il TextBuffer e' fondamentale perche' la TextView possa essere utile, ci sono due metodi per assegnarlo, in fase di creazione della TextView oppure in seguito, in entrambi i casi bisognera' prima costruire il TextBuffer

In fase di creazione della TextView procederemo in questo modo:

```
buffer = gtk.TextBuffer()
text = gtk.TextView(buffer)
```

Per assegnare il buffer successivamente utilizziamo il metodo **gtk.TextView.set_buffer(buffer)**

```
myBuffer = gtk.TextBuffer()
text = gtk.TextView()
text.set_buffer(myBuffer)
```

Rintracciare l'oggetto TextBuffer e' altrettanto semplice, utilizziamo il metodo **gtk.TextView.get_buffer(buffer)**

```
text.get_buffer(myBuffer)
```

Attenzione, questo metodo non restituisce il testo inserito ma l'**oggetto** TextBuffer

```
<gtk.TextBuffer object (GtkTextBuffer) at 0xb7ccacac>
```

14.2. gtk.TextBuffer()

TextBuffer e' il widget che contiene il testo da manipolare, il testo puo' essere ripartito in piu' TextView

La gerarchia

```
+-- GObject
```

```
+-- gtk.TextBuffer
```

Il costruttore

```
gtk.TextBuffer(texttagtable=None)
```

texttagtable: La TextTagTable è un contenitore di TextTag.

14.2.1. Contare linee e caratteri all'interno di un TextBuffer

TextBuffer ha due utilissimi metodi che permettono di conoscere in qualunque momento il numero di righe e il totale dei caratteri presenti al suo interno:

```
gtk.TextBuffer.get_line_count()
```

```
gtk.TextBuffer.get_char_count()
```

Example 14.2. Contare linee e caratteri

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

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

```
class MainWindow(gtk.Window):
```

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

```
        gtk.Window.__init__(self)
        self.connect("delete_event", self.doQuit)
        self.set_default_size(300, 200)
        self.set_border_width(5)
        self.buffer = gtk.TextBuffer()
        self.text = gtk.TextView(self.buffer)
        self.button = gtk.Button("conta righe e caratteri!")
        self.button.connect("clicked", self.doCount)
        self.countBar = gtk.Statusbar()
        self.countBar.set_has_resize_grip(gtk.FALSE)
        vbox = gtk.VBox(gtk.FALSE, 5)
        vbox.pack_start(self.text, gtk.TRUE, gtk.TRUE, 0)
        vbox.pack_start(self.button, gtk.FALSE, gtk.FALSE, 0)
        vbox.pack_start(self.countBar, gtk.FALSE, gtk.FALSE, 0)
        self.add(vbox)
        self.show_all()
```

```
    def doCount(self, widget, data=None):
```

```
        howManyLines = self.buffer.get_line_count()
        howManyChars = self.buffer.get_char_count()
        self.countBar.pop(0)
        self.countBar.push(0, 'total lines: %s chars: %s' % (str(howManyLines), str(howManyChars)))
```

```
    def doQuit(self, widget, data=None):
```

```
        gtk.main_quit()
```

```
    def main(self):
```

```
        gtk.main()
```

```
if __name__ == "__main__":
```

```
    go = MainWindow()
    go.main()
```

??? Inizializziamo una Statusbar in cui inserire i conteggi.

Utilizziamo il metodo `get_line_count()` per contare le linee.

Utilizziamo il metodo `get_char_count()` per contare i caratteri.

`buffer_get_lines_chars.py` ([../pyhtml/buffer_get_lines_chars.html](#))

Figure 14.2. Contare linee e caratteri



14.3. gtk.TextIter()

I `TextIter` sono dei segnalibri volatili, puntano ad una posizione tra due caratteri all'interno del testo. Sono considerati volatili perché in caso il `TextBuffer` venga modificato scompaiono. Sono fondamentali ad esempio nella ricerca del testo.

La gerarchia

```

+-- GObject.GBoxed
   +-- gtk.TextIter

```

Il costruttore

I `TextIter` non hanno un costruttore vero e proprio ma vengono creati con i metodi di `TextBuffer`

14.3.1. Catturare il `TextIter` alla posizione corrente

Per ottenere il `TextIter` alla posizione corrente è possibile utilizzare una combinazione dei tanti metodi del `TextBuffer`:

```

#!/usr/bin/env python

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

class MainWindow(gtk.Window):
    def __init__(self):
        gtk.Window.__init__(self)
        self.connect("destroy", lambda w: gtk.main_quit())
        self.set_default_size(300, 200)
        self.set_border_width(5)
        self.vbox = gtk.VBox()
        self.text = gtk.TextView()
        self.buffer = self.text.get_buffer()
        self.button = gtk.Button(None, gtk.STOCK_EXECUTE)
        self.button.connect("clicked", self.getPos)
        self.add(self.vbox)
        self.vbox.pack_start(self.text, gtk.TRUE, gtk.TRUE, 0)

```

PyGTK GUI programming

```
self.vbox.pack_start(self.button, gtk.FALSE, gtk.FALSE, 0)
self.show_all()

def getPos(self, widget, data=None):
    current = self.buffer.get_iter_at_mark(self.buffer.get_insert())
    print current

def main(self):
    gtk.main()

if __name__ == "__main__":
    go = MainWindow()
    go.main()
```

textView_get_iter_current.py (../pyhtml/textview_get_iter_current.html)

current corrispondera' quindi al TextIter alla posizione corrente

14.4. gtk.TextMark()

I TextMark a differenza dei TextIter sono dei segnalibri permanenti, ed anche loro puntano ad una posizione tra due caratteri all'interno del testo, anche in caso il TextBuffer venga modificato rimangono, sono gli strumenti ideali nell'implementazioni di comandi come undo.

La gerarchia

```
+-- GObject
+-- gtk.TextMark
```

Il costruttore

Come i TextIter anche i TextMark non hanno un costruttore vero e proprio ma vengono creati con i metodi di TextBuffer

14.4.1. Catturare il TextMark alla posizione corrente

Per ottenere il TextMark alla posizione corrente e' possibile utilizzare uno dei tanti metodi del TextBuffer:

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

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

class MainWindow(gtk.Window):
    def __init__(self):
        gtk.Window.__init__(self)
        self.connect("destroy", lambda w: gtk.main_quit())
        self.set_default_size(300, 200)
        self.set_border_width(5)
        self.vbox = gtk.VBox()
        self.text = gtk.TextView()
        self.buffer = self.text.get_buffer()
        self.button = gtk.Button(None, gtk.STOCK_EXECUTE)
        self.button.connect("clicked", self.getPos)
        self.add(self.vbox)
        self.vbox.pack_start(self.text, gtk.TRUE, gtk.TRUE, 0)
        self.vbox.pack_start(self.button, gtk.FALSE, gtk.FALSE, 0)
        self.show_all()
```

PyGTK GUI programming

```
def getPos(self, widget, data=None):
    current = self.buffer.get_mark("insert")
    print current

def main(self):
    gtk.main()

if __name__ == "__main__":
    go = MainWindow()
    go.main()
```

textView_get_mark_current.py (../pyhtml/textview_get_mark_current.html)

current corrispondera' quindi al TextMark alla posizione corrente

Esiste un altro metodo che richiede meno... "digitazione", basta sostituire il metodo `gtk.TextBuffer.get_mark(nome_mark)` con:

```
current = self.buffer.get_insert()
```

14.5. `gtk.TextTagTable()`

La `TextTagTable` e' un contenitore di `TextTag`, ovvero gli attributi da associare al testo.

La gerarchia

```
+-- GObject
   |-- gtk.TextTagTable
```

Il costruttore

```
gtk.TextTagTable()
```

14.6. `gtk.TextTag()`

I `TextTag` sono gli attributi da associare al testo.

La gerarchia

```
+-- GObject
   |-- gtk.TextTag
```

Il costruttore

```
gtk.TextTag(name=None)
```

name: Un nome da associare al `TextTag`.

Chapter 15. Look and feel

Poter personalizzare le GUI è un aspetto fondamentale nella scelta del toolkit da utilizzare.

Questo è uno degli aspetti che rendono molto interessante PyGTK, utilizzando questo toolkit abbiamo a disposizione una quantità di strumenti utili per applicare ai widget le nostre personalizzazioni, bisogna sempre tener presente che il window manager ha il controllo sugli stili impostati sull'utente, ma comunque e' possibile personalizzare tutta una serie di caratteristiche che andiamo a vedere.

Ci sono molti metodi per cambiare il look ai widget che utilizziamo nella creazione delle GUI, alcuni sono applicabili solo a widget con determinate caratteristiche, ad esempio quelli dotati di una window, altri metodi invece sono univoci e quindi applicabili solo ad un widget particolare.

15.1. Modificare il colore di background di una Window

Supponiamo ad esempio di voler cambiare il colore di background ad un widget Window, il metodo e' piuttosto semplice richiede soli 2 passaggi.

```
gtk.Widget.modify_bg(state, color)
```

state: indica lo stato di un widget, il valore puo' essere uno dei seguenti:

gtk.STATE_NORMAL: Questo come dice il nome stesso e' lo stato normale di un widget.

gtk.STATE_ACTIVE: Stato di un widget correntemente attivo, come puo' essere un bottone non premuto.

gtk.STATE_PRELIGHT: Stato che indica che il mouse è sopra di esso, il widget e' pronto per rispondere ad eventuali eventi generati dal mouse

gtk.STATE_SELECTED: Stato che indica il widget come selezionato.

gtk.STATE_INSENSITIVE: Stato che indica il widget come inattivo

color: e' il colore da assegnare al background, deve essere di tipo `gtk.gdk.color`.

Possiamo procedere riprendendo il codice utilizzato per la creazione della prima finestra e modificandolo come segue:

Example 15.1. Modificare il background di una Window

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

class FirstWin:
    def __init__(self):
        self.win = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.win.set_title('cambia il colore!')
        color = gtk.gdk.color_parse('#234fdb')
        self.win.modify_bg(gtk.STATE_NORMAL, color)
        self.win.show()

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

```
if __name__ == "__main__":
    first = FirstWin()
    first.main()
```

win_change_bg_color.py (../pyhtml/win_change_bg_color.html)

Il metodo **gtk.gdk.color_parse()** restituisce ed assegna alla variabile **color** il colore da utilizzare come background

il metodo **modify_bg(state, color)** assegna di conseguenza il colore allo sfondo della window

Figure 15.1. Cambiare il colore di background



Al metodo **color_parse** possono essere passati come parametri uno a scelta tra il nome di un colore tra quelli elencati nelle specifiche X11 (file rgb.txt) oppure il valore in esadecimale nella forma vista nell'esempio.

15.2. Modificare il colore di foreground di una Label

Supponiamo ora di voler cambiare il colore di foreground ad un widget Label, anche questo metodo come **modify_bg** e' piuttosto semplice e richiede 2 passaggi:

Example 15.2. Modificare il colore di una Label

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

class SecondWin:
    def __init__(self):
        self.win = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.win.connect("delete_event", self.delete_event)
        self.win.connect("destroy", self.destroy)
        self.win.set_title('cambia il colore!')
        self.label = gtk.Label('Just something')
        self.win.add(self.label)
        color = gtk.gdk.color_parse('#234fdb')
        self.label.modify_fg(gtk.STATE_NORMAL, color)
        self.win.show_all()

    def delete_event(self, widget, event, data=None):
        return gtk.FALSE

    def destroy(self, widget, data=None):
        return gtk.main_quit()

    def main(self):
        gtk.main()

if __name__ == "__main__":
```

```
second = SecondWin()
second.main()
```

label_change_fg_color.py (../pyhtml/label_change_fg_color.html)

Il metodo **gtk.gdk.color_parse()** restituisce ed assegna alla variabile **color** il colore da utilizzare come background

il metodo **modify_fg(state, color)** assegna di conseguenza il colore alla label

Figure 15.2. Cambiare il colore di foreground



15.3. Modificare il colore di fg di una Label all'interno di un RadioButton

Supponiamo ora di voler cambiare il colore di foreground di una Label che si trova all'interno di un altro widget. Il metodo precedente non funzionerebbe, quello che si puo' fare è cambiare il colore del cerchio all'interno del RadioButton

Example 15.3. Modificare il colore di un RadioButton

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

class SecondWin:
    def __init__(self):
        self.win = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.win.connect("delete_event", self.delete_event)
        self.win.connect("destroy", self.destroy)
        self.win.set_default_size(200,70)
        self.win.set_title('cambia il colore!')
        self.button = gtk.RadioButton(None, "Ecco la label")
        color = gtk.gdk.color_parse('#234fdb')
        self.button.modify_text(gtk.STATE_NORMAL, color)
        self.win.add(self.button)
        self.win.show_all()

    def delete_event(self, widget, event, data=None):
        return gtk.FALSE

    def destroy(self, widget, data=None):
        return gtk.main_quit()

    def main(self):
        gtk.main()

if __name__ == "__main__":
    second = SecondWin()
    second.main()
```

button_change_circle_fg_color.py (../pyhtml/button_change_circle_fg_color.html)

Creiamo un bottone di tipo RadioButton a cui cambiare il colore.

il metodo `modify_text(state, color)` assegna di conseguenza il colore al cerchietto del `RadioButton`

Figure 15.3. Cambiare il colore di foreground del cerchietto di un `RadioButton`



Vediamo ora il metodo per cambiare il colore della Label. Per farlo bisogna utilizzare i metodi per leggere l'attributo `child` del `RadioButton` e il metodo per scrivere l'attributo della Label.

Example 15.4. Modificare il colore della Label di un `RadioButton`

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

class SecondWin:
    def __init__(self):
        self.win = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.win.connect("delete_event", self.delete_event)
        self.win.connect("destroy", self.destroy)
        self.win.set_default_size(200,70)
        self.win.set_title('cambia il colore!')
        self.button = gtk.RadioButton(None, "<span>I am RED!</span>")
        label = self.button.child
        label.set_property("use-markup", gtk.TRUE)
        self.win.add(self.button)
        self.win.show_all()

    def delete_event(self, widget, event, data=None):
        return gtk.FALSE

    def destroy(self, widget, data=None):
        return gtk.main_quit()

    def main(self):
        gtk.main()

if __name__ == "__main__":
    second = SecondWin()
    second.main()
```

button_change_fg_color.py (../pyhtml/button_change_fg_color.html)

Creiamo un bottone di tipo `RadioButton` con una Label a cui cambiare il colore utilizzando il pango markup.

il metodo `child` legge l'attributo "child" del `RadioButton` e lo assegna alla variabile `label` con il metodo `set_property` settiamo la proprieta' "use-markup" a `TRUE`

Figure 15.4. Cambiare il colore di foreground della Label di un `RadioButton`



15.4. Creare un bottone con stock e testo personalizzato

Creiamo ora un bottone utilizzando uno `stock_item` ma cambiandone il testo predefinito con uno adatto alla nostra GUI.

Example 15.5. Modificare il testo di uno stock item

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

class SecondWin(gtk.Window):
    def __init__(self):
        gtk.Window.__init__(self, gtk.WINDOW_TOPLEVEL)
        self.connect("delete_event", self.delete_event)
        self.connect("destroy", self.destroy)
        self.set_title('new stock!')
        self.set_border_width(10)
        gtk.stock_add([(gtk.STOCK_QUIT, " Di qui si esce...", 0, 0, "")])
        self.button = gtk.Button(None, gtk.STOCK_QUIT)
        self.button.connect("clicked", self.destroy)
        self.add(self.button)
        self.show_all()

    def delete_event(self, widget, event, data=None):
        return gtk.FALSE

    def destroy(self, widget, data=None):
        return gtk.main_quit()

    def main(self):
        gtk.main()

if __name__ == "__main__":
    second = SecondWin()
    second.main()
```

new_stock.py (../pyhtml/new_stock.html)

Il metodo `gtk.stock_add`([stock, label, modifier, keyval, translation_domain])

Figure 15.5. Stock_item con testo personalizzato



Chapter 16. Riutilizzare il codice

16.1. Una finestra riutilizzabile

Come già precedentemente menzionato, utilizzare un linguaggio orientato agli oggetti porta numerosi vantaggi, capita molto spesso scrivendo grandi applicazioni che richiedono una GUI, che il coding diventi un esercizio di digitazione.

Se ci accorgessimo ad esempio che dei widget dovranno essere usati di frequente, invece di far pratica con il copia incolla... possiamo scrivere delle porzioni di codice come vedremo tra poco.

Supponiamo ad esempio di aver bisogno di una finestra a cui associare un titolo e delle dimensioni (che variano di volta in volta), in questo caso possiamo fare un primo file del tipo:

Example 16.1. La classe da riutilizzare

```
class MyWin(gtk.Window):
    def __init__(self, width, height, title):
        gtk.Window.__init__(self)
        self.set_title(title)
        self.set_default_size(width, height)

        self.connect("delete_event", self.delete_event)
        self.connect("destroy", self.destroy)

    def delete_event(self, widget, event, data=None):
        return gtk.FALSE

    def destroy(self, widget, data=None):
        return gtk.main_quit()
```

Analizzando il codice possiamo notare che i metodi utilizzati sono quelli già visti in precedenza, la novità consiste e scusate il gioco di parole, nel subclass della classe `gtk.Window`.

Quest'ultima come vedremo

Creiamo quindi il secondo file ed importiamo il primo come normalmente faremmo per un modulo.

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

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

class SecondWin:
    def __init__(self):
        self.win = first.MyWin(150, 300, "example")

        self.hbox = gtk.HBox(gtk.TRUE, 0)
        self.win.add(self.hbox)

        self.button = gtk.Button("Hello World")
        self.button.connect("clicked", self.hello, None)
        self.button.connect_object("clicked", gtk.Widget.destroy, self.win)

        self.button1 = gtk.Button("Hello")
        self.button1.connect("clicked", self.hello, None)
```

PyGTK GUI programming

```
self.button1.connect_object("clicked", gtk.Widget.destroy, self.win)

self.hbox.pack_start(self.button, gtk.TRUE, gtk.TRUE, 0)
self.hbox.pack_start(self.button1, gtk.TRUE, gtk.TRUE, 0)

self.win.show_all()

def hello(self, widget, data=None):
    print "Hello, World"

def main(self):
    gtk.main()

if __name__ == "__main__":
    second = SecondWin()
    second.main()
```

Le due linee in grassetto sono il cuore della nuova applicazione, con la prima linea abbiamo importato il primo file, la seconda e' il costruttore della nuova classe. Oltre a creare la nuova finestra con dimensioni e il titolo specificato nel costruttore, la finestra sara' gia' dotata degli eventi di chiusura, `delete_event` e `destroy`, non ci sara' quindi la necessita' di ridigitarli o copiarli nella nuova applicazione, comodo vero :)

Appendix A. Recensione in 5 minuti

Chapter 1. Introduzione

- 1.1. Primo approccio

Sviluppare applicazioni in maniera rapida ed efficace e' ormai diventato un must, i tempi per la realizzazione dei progetti sono sempre piu' stretti, al programmatore viene richiesto tutto e subito, per queste ragioni l'accoppiata Python-PyGTK puo' essere una delle scelte migliori.

- 1.2. Il toolkit PyGTK

Pygtk e' un set di moduli che permettono l'interfaccia tra Python e le librerie GTK, e' un toolkit object oriented, permette quindi la possibilita' di riusare lo stesso codice in piu' applicazioni, e' stato portato ai Python coders da James Henstridge in collaborazione con un team di developers volontari.

- 1.3. PyGTK e Glade

Glade, per chi non ne avesse mai sentito parlare, e' un utilissimo strumento che permette di organizzare una GUI in maniera visuale.

- 1.4. IDE o editor

Come per qualsiasi altro linguaggio, nello scrivere applicazioni con Python-PyGTK bisogna decidere quali tool di sviluppo utilizzare, generalmente per sviluppare semplici programmi un editor e' piu' che sufficiente, mentre per applicazioni piu' complesse dove magari puo' rendersi necessaria una fase di debug, e' meglio passare ad utilizzare una delle tante IDE (Integrated Development Enviroment) a disposizione.

- 1.5. Installazione

PyGTK puo' essere utilizzato indipendentemente dalla piattaforma, e' pienamente supportato da GNU/Linux e Windows, puo' essere utilizzato anche sul Mac installando un Xserver.

- 1.6. Supporto e help

La documentazione disponibile sulla rete e' quasi esclusivamente in inglese, sono disponibili sia per la consultazione online sia per il download un tutorial ed un reference manual, entrambi redatti da John Finlay alla home page www.pygtk.org.

Chapter 2. I Widget, le classi ed un primo esempio

- 2.1. I widget

Come gia' precedentemente menzionato, il toolkit PyGTK e' Object Oriented, i widget non sono altro che delle classi con dei metodi associati.

- 2.2. Un primo esempio di GUI

Passiamo ora ai fatti creando la prima finestra con PyGTK!!

- 2.3. Estendiamo l'esempio, un classico...Hello, World!

Tutti gli esempi che si rispettino hanno un bottone Hello, World, potremmo essere da meno...??

Chapter 3. Segnali, eventi e callbacks

PyGTK GUI programming

- 3.1. I segnali e la classe padre `gobject.GObject`

Chi si occupa di connettere i segnali e passarli alla funzione appropriata e' `gobject.GObject` con i suoi metodi.

- 3.2. Gli eventi

Oltre alla gestione dei segnali esiste anche la gestione degli eventi, la differenza fondamentale tra i due e' che i primi (segnali) sono emessi dai widget, i secondi sono emessi dal server X e dalle periferiche ad esso collegate.

Chapter 4. Gli attributi e le proprieta' dei widget

- 4.1. Gli attributi dei widget

Gli attributi dei widget sono fundamentalmente delle caratteristiche degli stessi e per lo piu' sono di sola lettura.

- 4.2. Le proprieta' dei widget

Le proprieta' dei widget appartengono all'oggetto `GObject`, e' possibile accedervi tramite dei metodi di quest'ultimo.

Chapter 5. Impacchettare i widget

- 5.1. Utilizzare i packing box

I packing box non sono altro che una serie di scatole invisibili in cui inserire i widget, possiamo utilizzare 2 tipi differenti di box, orizzontali `gtk.Hbox()` e verticale `gtk.Vbox`, entrambi discendono dal widget padre `gtk.Box`.

- 5.2. Utilizzare la packing table

Un altro metodo per posizionare gli oggetti all'interno della GUI e' quello di usare una tabella, questo strumento a volte puo' essere davvero indispensabile,

Chapter 6. Il widget contenitore, La finestra.

- 6.1. `gtk.Window()`

Nel capitolo precedente abbiamo gia' visto questo widget in azione, aggiungiamo ora dei nuovi contenuti sviluppando la nostra prima finestra.

Chapter 7. I bottoni

- 7.2. `gtk.Button()`

Dopo aver discusso del funzionamento di alcuni segnali e della costruzione del widget `Window` andiamo ad applicare le nozioni fin qui apprese, implementando con un bottone la nostra precedente applicazione. Creeremo inoltre una funzione di callback e applicheremo dei segnali.

- 7.3. `gtk.ToggleButton()`

Questi 3 bottoni derivano da `Button` sono infatti a lui molto simili, la differenza fondamentale e' quella di poter assumere due stati, normale ed attivo, hanno inoltre caratteristiche comuni, `ToggleButton` e' la base su cui sono costruiti, molte delle sue caratteristiche infatti valgono anche per i restanti due.

- 7.4. `gtk.CheckButton()`

Il secondo bottone ovvero `CheckButton` e' concettualmente molto simile al precedente.

- 7.5. `gtk.RadioButton()`

Il terzo bottone ovvero `RadioButton` non ha grandi differenze rispetto ai precedenti se utilizzato singolarmente, se invece vengono inseriti piu' `RadioButton` si formera' un gruppo. Il gruppo puo' comodamente essere usato per far scegliere allo user una o piu' opzioni all'interno della GUI

Chapter 8. Metodi, attributi, proprieta' e segnali delle classi widget, container e bin

- 8.1. La classe widget

La classe widget gerarchicamente e' forse la piu' importante, da essa infatti discendono tutti i widget intesi come oggetti con cui interagire, sono quindi esclusi oggetti come `TextBuffer` e `Tooltips`, discendono infatti da `GObject` il primo e da `gtk.Object` il secondo.

Chapter 9. Scendiamo di un livello, il wrapper GDK

- 9.1. `gtk.gdk.Window()`

Come detto in precedenza questa e' la base su cui sono costruiti la maggior parte dei widget, in pratica si tratta di una regione rettangolare dello schermo che puo' essere mostrata o nascosta, nella libreria Xlib queste funzioni vengono chiamate `mapping` e `unmapping`.

- 9.2. `gtk.gdk.Cursor()`

Ogni `gtk.gdk.Window` puo' essere dotata di un `gtk.gdk.Cursor` questo non e' altro che un'immagine bitmap che indica la posizione del mouse, per default il cursore utilizzato e' quello della parent `Window`

- 9.3. `gtk.gdk.Keymap()`

`gtk.gdk.Keymap` e' lo strumento principale con cui catturare la pressione dei tasti, traducendo il segnale che arriva dalla tastiera ad un valore interpretabile.

Chapter 10. Un ulteriore widget fondamentale, la Label

- 10.1. `gtk.Label()`

Un altro dei widget fondamentali utilizzati nelle GUI sono le `Label`, utilizzerò sempre il termine inglese in quanto la traduzione etichette sinceramente non mi pare appropriata :)

Chapter 11. Interagire con l'utente

- 11.1. `gtk.Entry()`

Interagire con l'utente e' sicuramente un compito fondamentale per una GUI, le `Entry` sono lo strumento principale per svolgere questo compito.

Chapter 12. Messaggi di dialogo

- 12.1. `gtk.Dialog()`

Un altro modo per interagire con l'utente e' quello di far apparire dei messaggi di dialogo. Un esempio classico e' la finestra di popup che chiede la conferma di chiudere l'applicazione.

- 12.2. `gtk.MessageDialog()`

`MessageDialog` come `Dialog` e' un widget relativamente semplice e potrebbe tranquillamente essere scritto utilizzando quest'ultimo, e' utile perche' in ogni caso permette di risparmiare un po' di... "digitazione"

Chapter 13. I tool widget e la statusbar

- 13.1. `gtk.Toolbar()`

La `Toolbar` e' un widget utile nel caso si voglia sostituire o implementare un menu un maniera visuale. Gli oggetti nella `Toolbar` infatti sono di facile individuazione anche grazie all'applicazione di `Tooltips` che vedremo in seguito.

- 13.2. `gtk.Tooltips()`

I `Tooltips` sono molto utili nella descrizione di altri widget presenti all'interno delle applicazioni

- 13.3. `gtk.Statusbar()`

La `Statusbar` e' utilizzata solitamente per mettere in display informazioni utili, quali ad esempio il nome di un file ma anche il numero di righe e caratteri in un editor di testi, vedi ad esempio il Chapter 14, *Manipolare il testo*.

Chapter 14. Manipolare il testo

- 14.1. `gtk.TextView()`

`TextView` e' il widget che avra' al suo interno il testo da manipolare, a partire dalla versione 2.0 di GTK+ il testo e' codificato UTF-8. Questo non e' l'unico strumento che ci permettera' di manipolare i testi, via via scopriremo anche gli altri.

- 14.2. `gtk.TextBuffer()`

`TextBuffer` e' il widget che contiene il testo da manipolare, il testo puo' essere ripartito in piu' `TextView`

- 14.3. `gtk.TextIter()`

I `TextIter` sono dei segnalibri volatili, puntano ad una posizione tra due caratteri all'interno del testo. Sono considerati volatili perche' in caso il `TextBuffer` venga modificato scompaiono. Sono fondamentali ad esempio nella ricerca del testo.

- 14.4. `gtk.TextMark()`

I `TextMark` a differenza dei `TextIter` sono dei segnalibri permanenti, ed anche loro puntano ad una posizione tra due caratteri all'interno del testo, anche in caso il `TextBuffer` venga modificato rimangono, sono gli strumenti ideali nell'implementazioni di comandi come `undo`.

- 14.5. `gtk.TextTagTable()`

La `TextTagTable` e' un contenitore di `TextTag`, ovvero gli attributi da associare al testo.

- 14.6. `gtk.TextTag()`

I `TextTag` sono gli attributi da associare al testo.

Chapter 15. Look and feel

- 15.1. Modificare il colore di background di una Window

Supponiamo ad esempio di voler cambiare il colore di background ad un widget Window, il metodo e' piuttosto semplice richiede soli 2 passaggi.

- 15.2. Modificare il colore di foreground di una Label

Supponiamo ora di voler cambiare il colore di foreground ad un widget Label, anche questo metodo come `modify_bg` e' piuttosto semplice e richiede 2 passaggi:

- 15.3. Modificare il colore di fg di una Label all'interno di un RadioButton

Supponiamo ora di voler cambiare il colore di foreground di una Label che si trova all'interno di un altro widget. Il metodo precedente non funzionerebbe, quello che si puo' fare è cambiare il colore del cerchio all'interno del RadioButton

- 15.4. Creare un bottone con stock e testo personalizzato

Creiamo ora un bottone utilizzando uno `stock_item` ma cambiandone il testo predefinito con uno adatto alla nostra GUI.

Chapter 16. Riutilizzare il codice

- 16.1. Una finestra riutilizzabile

Come gia' precedentemente menzionato, utilizzare un linguaggio orientato agli oggetti porta numerosi vantaggi, capita molto spesso scrivendo grandi applicazioni che richiedono una GUI, che il coding diventi un esercizio di digitazione.

Appendix B. Tips and tricks

Chapter 1. Introduzione

Chapter 2. I Widget, le classi ed un primo esempio

Chapter 3. Segnali, eventi e callbacks

Chapter 4. Gli attributi e le proprieta' dei widget

Chapter 5. Impacchettare i widget

Chapter 6. Il widget contenitore, La finestra.

- 6.1.3. I widget transitori

Passando come valore **None** sara' rimossa un'eventuale transitorietà'.

- 6.1.6. Muovere la finestra

E' bene precisare che la posizione indicata non sara' quella della finestra in relazione allo schermo, bensì e' il punto di partenza delle coordinate che possono essere passate da altri metodi come vedremo piu' avanti, in pratica il punto 0.0 di un ipotetico piano cartesiano.

La direzione del piano cartesiano sara' dall'alto verso il basso e da sinistra verso destra

- 6.1.10. Segnali ed eventi della Window

Solo alla pressione dei tasti Enter e Return il segnale sara' emesso dalla window, cio' nonostante l'attivazione dei Button reagira' anche al click del mouse o della barra spaziatrice.

Questo segnale invia la direzione corrispondente al tasto premuto, da notare quindi il parametro aggiuntivo alla funzione di callback.

Chapter 7. I bottoni

- 7.3.2. I segnali di ToggleButton

Ad ogni modo non ha molto senso connettere un segnale come "clicked" ad un ToggleButton

- 7.5.3. I segnali di RadioButton

Anche per il RadioButton non ha molto senso connettere un segnale come "clicked".

Chapter 8. Metodi, attributi, proprieta' e segnali delle classi widget, container e bin

- 8.1.1. Impadronirsi del controllo

Ovviamente ogni volta che incontriamo la forma standard del metodo trattato, per applicarla ai nostri programmi dovremmo sostituire `gtk.Widget` con il nome dell' oggetto che stiamo utilizzando, nel caso precedente bisogna utilizzare:

```
self.button1.grab_remove()
```

Chapter 9. Scendiamo di un livello, il wrapper GDK

Chapter 10. Un ulteriore widget fondamentale, la Label

- 10.1.1. Inserire il testo nella Label

Attenzione che se il mnemonic accelerator non e' associato a nessuna funzione, un `GtkWarning` sara' restituito:

PyGTK GUI programming

```
GtkWarning: Couldn't find a target for a mnemonic activation. gtk.main()
```

- 10.1.6. Operazioni di selezione in una Label

Questo metodo non puo' essere eseguito in fase di creazione della Label ma quest'ultima deve gia' essere in display. Diversamente un GtkWarnign sara' restituito e il testo non sara' inserito nella clipboard.

```
GtkWarning: file gtkwidget.c: line 7344 (gtk_widget_get_clipboard): assertion `gtk_widge
GtkWarning: file gtkclipboard.c: line 561 (gtk_clipboard_set_with_owner): assertion `cli
```

Impostando il valore `-1` alla selezione, questa andra' fino alla fine della Label

Chapter 11. Interagire con l'utente

- 11.1. `gtk.Entry()`

Il range del numero di caratteri varia da 0 a 65536, settare il valore a 0 significa non assegnare un valore massimo

- 11.1.4. Impostare il numero massimo di caratteri visualizzabili

Come il widget verra' visualizzato dipende anche dal metodo utilizzato per inserirlo nella GUI.

- 11.1.5. Modificare l'allineamento del testo

Il valore deve essere una frazione di intero espresso in numero decimale, il valore 1 allineera' il testo verso destra.

Chapter 12. Messaggi di dialogo

Chapter 13. I tool widget e la statusbar

Chapter 14. Manipolare il testo

- 14.1.2. Associare e recuperare un `TextBuffer` dalla `TextView`

Attenzione, questo metodo non restituisce il testo inserito ma l'**oggetto** `TextBuffer`

```
<gtk.TextBuffer object (GtkTextBuffer) at 0xb7ccacac>
```

- 14.4.1. Catturare il `TextMark` alla posizione corrente

Esiste un altro metodo che richiede meno... "digitazione", basta sostituire il metodo `gtk.TextBuffer.get_mark(nome_mark)` con:

```
current = self.buffer.get_insert()
```

Chapter 15. Look and feel

- 15.1. Modificare il colore di background di una Window

Al metodo **color_parse** possono essere passati come parametri uno a scelta tra il nome di un colore tra quelli elencati nelle specifiche X11 (file `rgb.txt`) oppure il valore in esadecimale nella forma vista nell'esempio.

Chapter 16. Riutilizzare il codice

Appendix C. Lista degli esempi

Chapter 1. Introduzione

Chapter 2. I Widget, le classi ed un primo esempio

- 2.2. Un primo esempio di GUI
 - ◆ Example 2.1. La creazione di una Window
- 2.3. Estendiamo l'esempio, un classico...Hello, World!
 - ◆ Example 2.2. Un bottone Hello, World!

Chapter 3. Segnali, eventi e callbacks

Chapter 4. Gli attributi e le proprieta' dei widget

Chapter 5. Impacchettare i widget

Chapter 6. Il widget contenitore, La finestra.

- 6.1.6. Muovere la finestra
 - ◆ Example 6.1. Spostare una finestra con le coordinate

Chapter 7. I bottoni

- 7.2. `gtk.Button()`
 - ◆ Example 7.1. Primo esempio di Button
- 7.2.3. I segnali emessi da Button
 - ◆ Example 7.2. Emettere un segnale

Chapter 8. Metodi, attributi, proprieta' e segnali delle classi widget, container e bin

- 8.1.1. Impadronirsi del controllo
 - ◆ Example 8.1.

Chapter 9. Scendiamo di un livello, il wrapper GDK

- 9.1.1. Catturare una `gtk.gdk.Window`
 - ◆ Example 9.1. Catturare una `gtk.gdk.Window`
- 9.2.1. Cambiare il cursore di una `gtk.gdk.Window`
 - ◆ Example 9.2. Cambiare il cursore ad un widget
- 9.3.1. Catturare una `gtk.gdk.Keymap`
 - ◆ Example 9.3. Catturare una `gtk.gdk.Keymap`

Chapter 10. Un ulteriore widget fondamentale, la Label

- 10.1. `gtk.Label()`
 - ◆ Example 10.1. Primo esempio di Label
- 10.1.1. Inserire il testo nella Label
 - ◆ Example 10.2. Inserire o cambiare il testo nella Label
- 10.1.3. Associare un mnemonic accelerator ad un widget esterno
 - ◆ Example 10.3.
- 10.1.5. Utilizzare il Pango markup in una Label
 - ◆ Example 10.4.
- 10.1.6. Operazioni di selezione in una Label
 - ◆ Example 10.5. Selezionare il testo

Chapter 11. Interagire con l'utente

- 11.1. `gtk.Entry()`
 - ◆ Example 11.1. Primo esempio di Entry
- 11.1.1. Leggere e scrivere il testo
 - ◆ Example 11.2. Prendere l'input e assegnarlo ad una variabile
- 11.1.2. Inserire il testo nella Entry
 - ◆ Example 11.3. Inserire il testo nella Entry
 - ◆ Example 11.4. Cambiare il testo nella Entry
- 11.1.3. Rendere invisibile il testo
 - ◆ Example 11.5. Rendere invisibile il testo
 - ◆ Example 11.6. Cambiare il carattere di default
- 11.1.4. Impostare il numero massimo di caratteri visualizzabili
 - ◆ Example 11.7. Numero di caratteri in display
- 11.1.5. Modificare l'allineamento del testo
 - ◆ Example 11.8.

Chapter 12. Messaggi di dialogo

- 12.1.1. Un primo esempio di Dialog
 - ◆ Example 12.1. Sicuro di voler uscire?
- 12.2.1. Un primo esempio di `MessageDialog`
 - ◆ Example 12.2. Delle utili info
- 12.2.2. Utilizzare il markup nella Label all'interno di un `MessageDialog`
 - ◆ Example 12.3. Catturare la Label

Chapter 13. I tool widget e la statusbar

- 13.1.1. Inserire un item nella Toolbar

PyGTK GUI programming

- ◆ Example 13.1. Inserire i bottoni nella Toolbar
- 13.2.2. Assegnare un Tooltips ad un widget;
 - ◆ Example 13.2. Associare un Tooltips ad un bottone.

Chapter 14. Manipolare il testo

- 14.1.1. Un primo esempio di TextView
 - ◆ Example 14.1. La creazione di una TextView
- 14.2.1. Contare linee e caratteri all'interno di un TextBuffer
 - ◆ Example 14.2. Contare linee e caratteri

Chapter 15. Look and feel

- 15.1. Modificare il colore di background di una Window
 - ◆ Example 15.1. Modificare il background di una Window
- 15.2. Modificare il colore di foreground di una Label
 - ◆ Example 15.2. Modificare il colore di una Label
- 15.3. Modificare il colore di fg di una Label all'interno di un RadioButton
 - ◆ Example 15.3. Modificare il colore di un RadioButton
 - ◆ Example 15.4. Modificare il colore della Label di un RadioButton
- 15.4. Creare un bottone con stock e testo personalizzato
 - ◆ Example 15.5. Modificare il testo di uno stock item

Chapter 16. Riutilizzare il codice

- 16.1. Una finestra riutilizzabile
 - ◆ Example 16.1. La classe da riutilizzare

Appendix D. Storia delle revisioni

Revision History	
Revision 2.0.0	02-01-2005
<ul style="list-style-type: none">• Completa ristrutturazione del libro, anno nuovo vita nuova....	
Revision 1.4.0	20-12-2004
<ul style="list-style-type: none">• iniziato il Chapter 14, <i>Manipolare il testo</i>.• aggiunte nuove parti in ???.• aggiunte nuove parti in Chapter 15, <i>Look and feel</i>.• aggiunte nuove parti nel Chapter 3, <i>Segnali, eventi e callbacks</i>.• Riorganizzata la struttura dei capitoli e delle revisioni• Sistemati dei link non funzionanti• Rimodificato il css	
Revision 1.3.0	08-12-2004
<ul style="list-style-type: none">• Aggiunto il Chapter 13, <i>I tool widget e la statusbar</i>.	
Revision 1.2.1	28-11-2004
<ul style="list-style-type: none">• Aggiunte due sezioni al Chapter 10, <i>Un ulteriore widget fondamentale, la Label</i>.• Sistemati dei link non funzionanti• Corretti alcuni errori di battitura	
Revision 1.2.0	27-11-2004
<ul style="list-style-type: none">• Aggiunto Chapter 11, <i>Interagire con l'utente</i>.• Aggiunto Chapter 12, <i>Messaggi di dialogo</i>.• Sistemati dei link non funzionanti• Corretti alcuni errori di battitura• Modificato il css	
Revision 1.1.0	25-11-2004
<ul style="list-style-type: none">• Corretti degli errori nel Chapter 1, <i>Introduzione</i>.• Aggiunto Chapter 15, <i>Look and feel</i>.• Sistemati alcuni link non funzionanti	
Revision 1.0.0	23-11-2004
<ul style="list-style-type: none">• Pubblicazione iniziale	

Appendix E. A proposito del libro

Questo libro e' in fase di scrittura con DocBook XML (<http://www.oasis-open.org/docbook/>) utilizzando come editor Anjuta (<http://anjuta.sourceforge.net/>), e convertito in HTML utilizzando xslt (<http://www.w3.org/TR/xslt>) partendo dai fogli di stile (e si vede...) utilizzati da Mark Pilgrim per la scrittura di dive into Python (<http://diveintopython.org/>) da qui e' stato convertito in PDF utilizzando HTMLDoc (<http://www.easysw.com/htmldoc/>), e sto cercando di portarlo a plain text utilizzando w3m (<http://ei5nazha.yz.yamagata-u.ac.jp/~aito/w3m/eng/>).

Appendix F. GNU Free Documentation License

Version 1.1, March 2000

Copyright (C) 2000 Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

F.0. Preamble

The purpose of this License is to make a manual, textbook, or other written document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondly, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

F.1. Applicability and definitions

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by

readers is not Transparent. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

F.2. Verbatim copying

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

F.3. Copying in quantity

If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

F.4. Modifications

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3

above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).
- C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
- D. Preserve all the copyright notices of the Document.
- E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- H. Include an unaltered copy of this License.
 - I. Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
 - J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- K. In any section entitled "Acknowledgements" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- M. Delete any section entitled "Endorsements". Such a section may not be included in the Modified Version.
- N. Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

F.5. Combining documents

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any sections entitled "Acknowledgements", and any sections entitled "Dedications". You must delete all sections entitled "Endorsements."

F.6. Collections of documents

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

F.7. Aggregation with independent works

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

F.8. Translation

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

F.9. Termination

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

F.10. Future revisions of this license

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <http://www.gnu.org/copyleft/> (<http://www.gnu.org/copyleft/>).

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

F.11. How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright (c) YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have no Invariant Sections, write "with no Invariant Sections" instead of saying which ones are invariant. If you have no Front-Cover Texts, write "no Front-Cover Texts" instead of "Front-Cover Texts being LIST"; likewise for Back-Cover Texts.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Appendix G. Python license

G.A. History of the software

Python was created in the early 1990s by Guido van Rossum at Stichting Mathematisch Centrum (CWI) in the Netherlands as a successor of a language called ABC. Guido is Python's principal author, although it includes many contributions from others. The last version released from CWI was Python 1.2. In 1995, Guido continued his work on Python at the Corporation for National Research Initiatives (CNRI) in Reston, Virginia where he released several versions of the software. Python 1.6 was the last of the versions released by CNRI. In 2000, Guido and the Python core development team moved to BeOpen.com to form the BeOpen PythonLabs team. Python 2.0 was the first and only release from BeOpen.com.

Following the release of Python 1.6, and after Guido van Rossum left CNRI to work with commercial software developers, it became clear that the ability to use Python with software available under the GNU Public License (GPL) was very desirable. CNRI and the Free Software Foundation (FSF) interacted to develop enabling wording changes to the Python license. Python 1.6.1 is essentially the same as Python 1.6, with a few minor bug fixes, and with a different license that enables later versions to be GPL-compatible. Python 2.1 is a derivative work of Python 1.6.1, as well as of Python 2.0.

After Python 2.0 was released by BeOpen.com, Guido van Rossum and the other PythonLabs developers joined Digital Creations. All intellectual property added from this point on, starting with Python 2.1 and its alpha and beta releases, is owned by the Python Software Foundation (PSF), a non-profit modeled after the Apache Software Foundation. See <http://www.python.org/psf/> for more information about the PSF.

Thanks to the many outside volunteers who have worked under Guido's direction to make these releases possible.

G.B. Terms and conditions for accessing or otherwise using Python

G.B.1. PSF license agreement

1. This LICENSE AGREEMENT is between the Python Software Foundation ("PSF"), and the Individual or Organization ("Licensee") accessing and otherwise using Python 2.1.1 software in source or binary form and its associated documentation.
2. Subject to the terms and conditions of this License Agreement, PSF hereby grants Licensee a nonexclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use Python 2.1.1 alone or in any derivative version, provided, however, that PSF's License Agreement and PSF's notice of copyright, i.e., "Copyright (c) 2001 Python Software Foundation; All Rights Reserved" are retained in Python 2.1.1 alone or in any derivative version prepared by Licensee.
3. In the event Licensee prepares a derivative work that is based on or incorporates Python 2.1.1 or any part thereof, and wants to make the derivative work available to others as provided herein, then Licensee hereby agrees to include in any such work a brief summary of the changes made to Python 2.1.1.
4. PSF is making Python 2.1.1 available to Licensee on an "AS IS" basis. PSF MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, PSF MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF PYTHON 2.1.1 WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.
5. PSF SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON 2.1.1 FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT

OF MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON 2.1.1, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

6. This License Agreement will automatically terminate upon a material breach of its terms and conditions.
7. Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between PSF and Licensee. This License Agreement does not grant permission to use PSF trademarks or trade name in a trademark sense to endorse or promote products or services of Licensee, or any third party.
8. By copying, installing or otherwise using Python 2.1.1, Licensee agrees to be bound by the terms and conditions of this License Agreement.

G.B.2. BeOpen Python open source license agreement version 1

1. This LICENSE AGREEMENT is between BeOpen.com ("BeOpen"), having an office at 160 Saratoga Avenue, Santa Clara, CA 95051, and the Individual or Organization ("Licensee") accessing and otherwise using this software in source or binary form and its associated documentation ("the Software").
2. Subject to the terms and conditions of this BeOpen Python License Agreement, BeOpen hereby grants Licensee a non-exclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use the Software alone or in any derivative version, provided, however, that the BeOpen Python License is retained in the Software, alone or in any derivative version prepared by Licensee.
3. BeOpen is making the Software available to Licensee on an "AS IS" basis. BEOPEN MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, BEOPEN MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF THE SOFTWARE WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.
4. BEOPEN SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF THE SOFTWARE FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THE SOFTWARE, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.
5. This License Agreement will automatically terminate upon a material breach of its terms and conditions.
6. This License Agreement shall be governed by and interpreted in all respects by the law of the State of California, excluding conflict of law provisions. Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between BeOpen and Licensee. This License Agreement does not grant permission to use BeOpen trademarks or trade names in a trademark sense to endorse or promote products or services of Licensee, or any third party. As an exception, the "BeOpen Python" logos available at <http://www.pythonlabs.com/logos.html> may be used according to the permissions granted on that web page.
7. By copying, installing or otherwise using the software, Licensee agrees to be bound by the terms and conditions of this License Agreement.

G.B.3. CNRI open source GPL-compatible license agreement

1. This LICENSE AGREEMENT is between the Corporation for National Research Initiatives, having an office at 1895 Preston White Drive, Reston, VA 20191 ("CNRI"), and the Individual or Organization ("Licensee") accessing and otherwise using Python 1.6.1 software in source or binary form and its associated documentation.
2. Subject to the terms and conditions of this License Agreement, CNRI hereby grants Licensee a nonexclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use Python 1.6.1 alone or in any derivative version, provided, however, that CNRI's License Agreement and CNRI's notice of

copyright, i.e., "Copyright (c) 1995–2001 Corporation for National Research Initiatives; All Rights Reserved" are retained in Python 1.6.1 alone or in any derivative version prepared by Licensee. Alternately, in lieu of CNRI's License Agreement, Licensee may substitute the following text (omitting the quotes): "Python 1.6.1 is made available subject to the terms and conditions in CNRI's License Agreement. This Agreement together with Python 1.6.1 may be located on the Internet using the following unique, persistent identifier (known as a handle): 1895.22/1013. This Agreement may also be obtained from a proxy server on the Internet using the following URL: <http://hdl.handle.net/1895.22/1013>".

3. In the event Licensee prepares a derivative work that is based on or incorporates Python 1.6.1 or any part thereof, and wants to make the derivative work available to others as provided herein, then Licensee hereby agrees to include in any such work a brief summary of the changes made to Python 1.6.1.
4. CNRI is making Python 1.6.1 available to Licensee on an "AS IS" basis. CNRI MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, CNRI MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF PYTHON 1.6.1 WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.
5. CNRI SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON 1.6.1 FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON 1.6.1, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.
6. This License Agreement will automatically terminate upon a material breach of its terms and conditions.
7. This License Agreement shall be governed by the federal intellectual property law of the United States, including without limitation the federal copyright law, and, to the extent such U.S. federal law does not apply, by the law of the Commonwealth of Virginia, excluding Virginia's conflict of law provisions. Notwithstanding the foregoing, with regard to derivative works based on Python 1.6.1 that incorporate non-separable material that was previously distributed under the GNU General Public License (GPL), the law of the Commonwealth of Virginia shall govern this License Agreement only as to issues arising under or with respect to Paragraphs 4, 5, and 7 of this License Agreement. Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between CNRI and Licensee. This License Agreement does not grant permission to use CNRI trademarks or trade name in a trademark sense to endorse or promote products or services of Licensee, or any third party.
8. By clicking on the "ACCEPT" button where indicated, or by copying, installing or otherwise using Python 1.6.1, Licensee agrees to be bound by the terms and conditions of this License Agreement.

G.B.4. CWI permissions statement and disclaimer

Copyright (c) 1991 – 1995, Stichting Mathematisch Centrum Amsterdam, The Netherlands.
All rights reserved.

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of Stichting Mathematisch Centrum or CWI not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission.

STICHTING MATHEMATISCH CENTRUM DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT SHALL STICHTING MATHEMATISCH CENTRUM BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE

USE OR PERFORMANCE OF THIS SOFTWARE.