GUI in Python

Graphical User Interface: pro e contro

- Pro
  - Interfaccia intuitiva per l'utente meno esperto
  - Si presta particolarmente bene all'avvio di operazioni multithreaded
- Contro
  - È molto difficile fornire un insieme di operazioni grafiche cui corrisponda un flusso di operazioni efficiente da parte dell'utente
  - È molto difficile automatizzare le operazioni

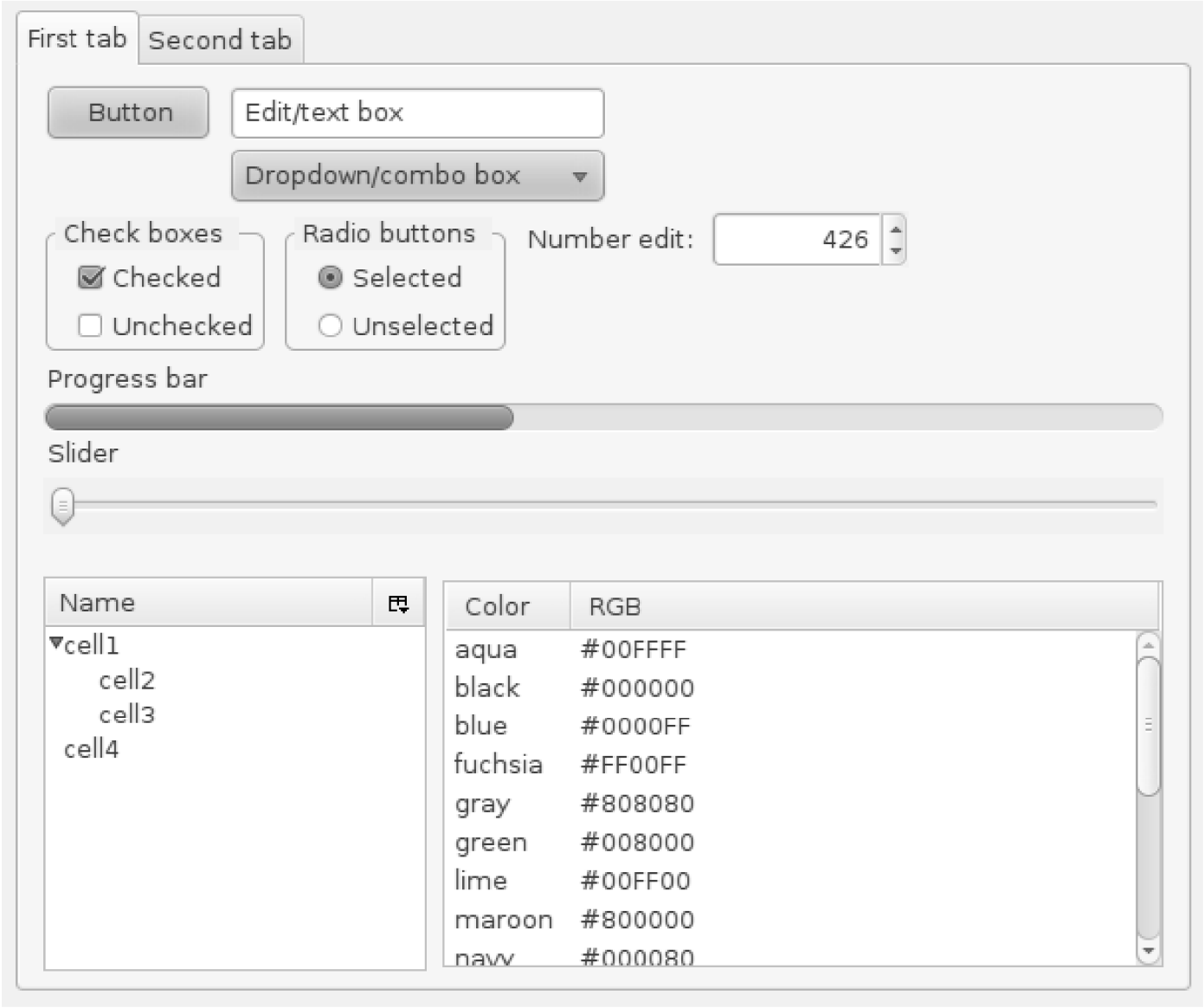
Graphical User Interface: sfide

- Ad ogni istante, l'utente può scegliere una fra tantissime operazioni disponibili
  - Pressione di un carattere, shortcut, click su un bottone, resizing di una finestra, etc.
  - Queste operazioni vanno tutte gestite
- L'applicazione deve essere progettata per eseguire in una finestra di dimensione variabile
  - Problemi di progettazione del layout grafico
  - Meccanismo per il ridisegno automatico del layout grafico in seguito a resizing

Organizzazione gerarchica

- Ciascun componente di una applicazione grafica prende il nome di widget
  - Bottoni, finestre, menu, liste, combo, aree di testo
- Un widget può "contenere" altri widget (si parla di composite widget)
- L'insieme dei widget è organizzato secondo una gerarchia
  - Dal punto di vista della programmazione: gerarchia ad oggetti
  - Dal punto di vista del layout grafico: un widget "contiene" visivamente altri oggetti

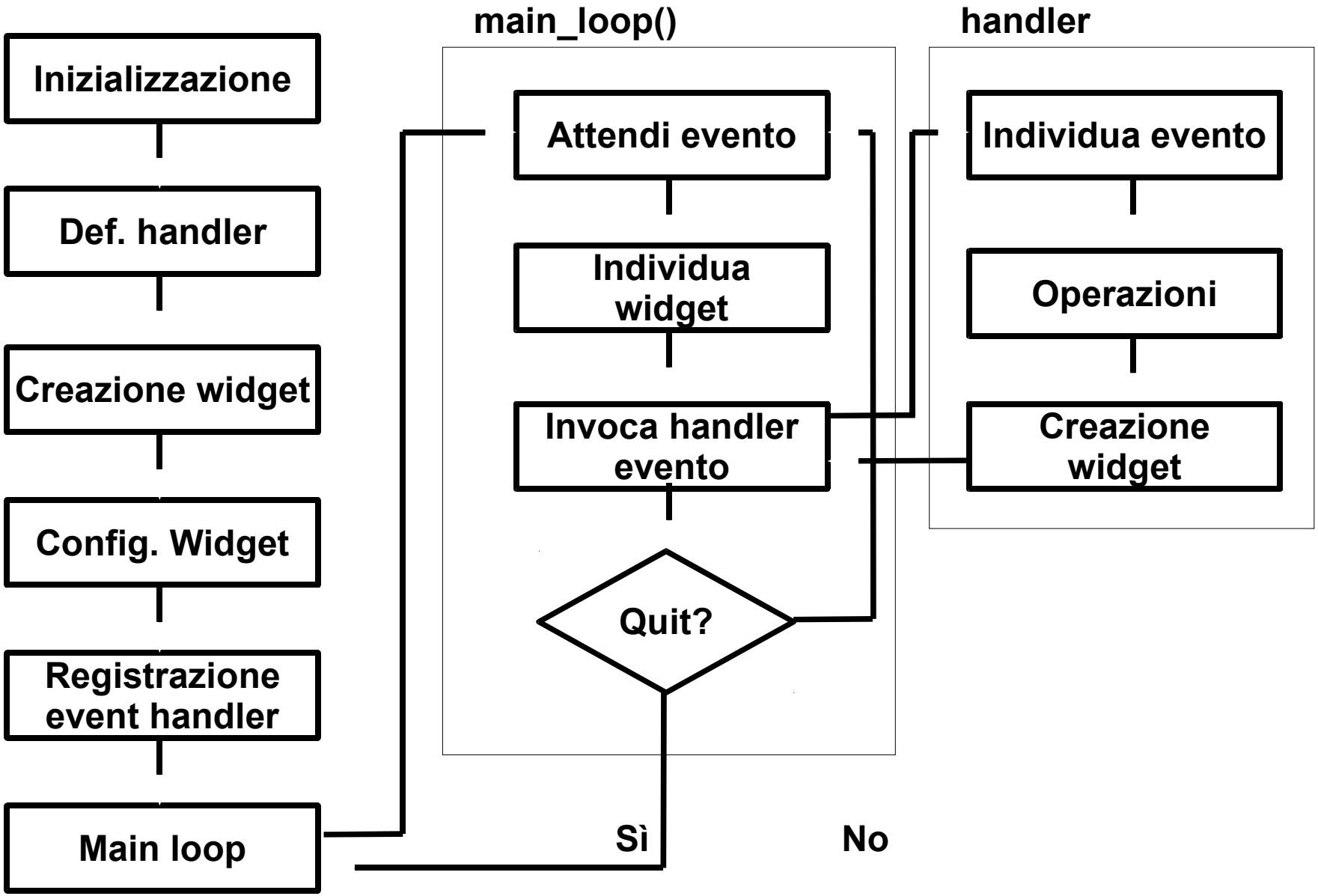
Organizzazione gerarchica



Paradigma ad eventi

- La realizzazione di una interfaccia grafica ben si sposa con il paradigma di programmazione ad eventi
- Si definiscono eventi le azioni possibili su un widget
  - Pressione di bottone, spostamento della finestra
- A ciascun evento può essere associata una funzione (handler, callback) che verrà invocata al verificarsi dell'evento
- L'applicazione
  - definisce i widget
  - associa i callback agli eventi
  - esegue un ciclo infinito (main loop) in attesa degli eventi

Paradigma ad eventi



I diversi widget toolkit

- Python offre binding per i widget toolkit più comuni
- Gimp Toolkit (GTK+)
  - Il toolkit di GNOME, usato in Debian ed Ubuntu
- Qt
  - Il toolkit di KDE
- Tk
  - Il toolkit del linguaggio TCL
- FLTK
- In questa lezione, esamineremo più a fondo il toolkit GTK+

GTK+: che cosa è

- GTK+: acronimo di GIMP ToolKit
- GIMP è lo GNU Image Manipulation Program, il "Photoshop" della GNU
- Storia
  - 1998: GTK+ nasce come rimpiazzo di Motif (il toolkit adottato in origine)
  - 2002: GTK+ v2 (noto anche come GTK2): usato in GNOME2
  - 2011: GTK+ v3 (noto anche come GTK3): usato in GNOME3 e Unity

GTK+: librerie componenti

- Glib: strutture dati avanzate, thread, sincronizzazione, message passing, logging
- Gobject: sistema ad oggetti portabile fra diversi linguaggi, accessibile attraverso una API
- Pango: rendering del testo, gestione dei font
- ATK: supporto per la accessibilità
- GDK: wrapper alle funzionalità di disegno e di gestione delle finestre offerte dal sottosistema grafico
- GTK: interfaccia di programmazione ad eventi
- Cairo: supporto per la grafica vettoriale

Riassumendo

- GTK+ è
  - un set di librerie per la costruzione di interfacce grafiche complesse
  - un'interfaccia verso le funzioni di basso livello del sistema operativo
  - uno strumento per il disegno vettoriale
- In questa lezione, esploreremo il primo punto (ossia, la libreria GTK)
- Gli esempi sono presi da http://www.pygtk.org/pygtk2tutorial/index.html

Portabilità

- Multipiattaforma
  - Unix (X.org)
  - Windows
  - MacOS X
- Multilinguaggio
  - C
  - C++
  - Python
  - Ruby
  - Java
  - Perl
  - PHP

Un primissimo esempio

- ESEMPI: base.py

- **Creazione di una finestra 200x200 (e nient'altro)**
  - **Illustra i metodi utilizzati in uno scheletro di applicazione grafica**
- **L'applicazione base.py si suddivide in tre parti distinte**
  - **Caricamento ed inizializzazione dei moduli**
  - **Creazione di una classe contenente il codice dell'applicazione grafica (widget, handler)**
  - **Esecuzione del codice dell'applicazione grafica (convenzione: all'interno di un metodo main() della classe)**

Un primissimo esempio

- ESEMPI: base.py

- **Caricamento ed inizializzazione dei moduli**

```
#!/usr/bin/python

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

Modulo di interfaccia con le librerie condivise di GTK+.

Si richiede l'uso di GTK2.

API ad eventi per la creazione di widget grafici (usa pygtk).

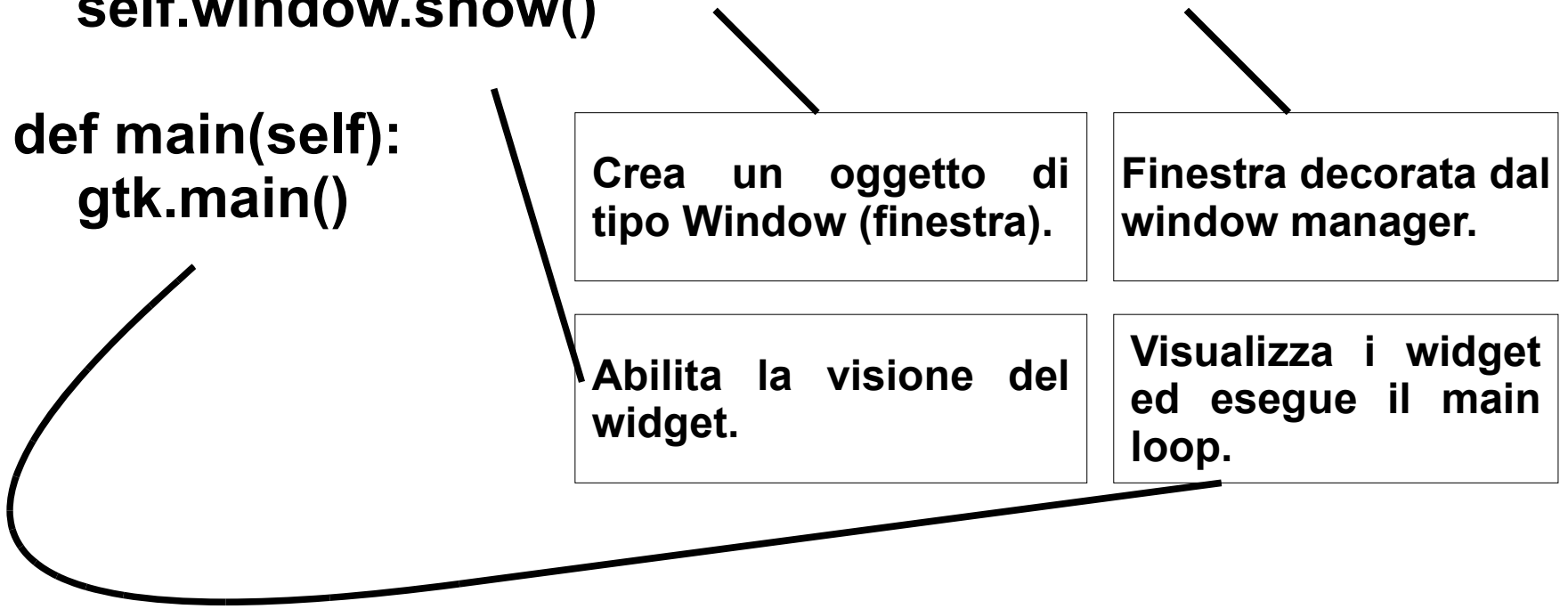
Un primissimo esempio

- ESEMPI: base.py

- **Creazione di una classe contenente il codice**

```
class Base:
    def __init__(self):
        self.window = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.window.show()

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



Un primissimo esempio

- ESEMPI: base.py

- **Esecuzione del codice dell'applicazione grafica**

```
if __name__ == “__main__”:
    base = Base()
    base.main()
```

Se il codice è invocato da linea di comando (script)...

… istanzia una classe Base ed esegue il codice della applicazione grafica.

Primi widget e gestione eventi

- ESEMPI: helloworld.py

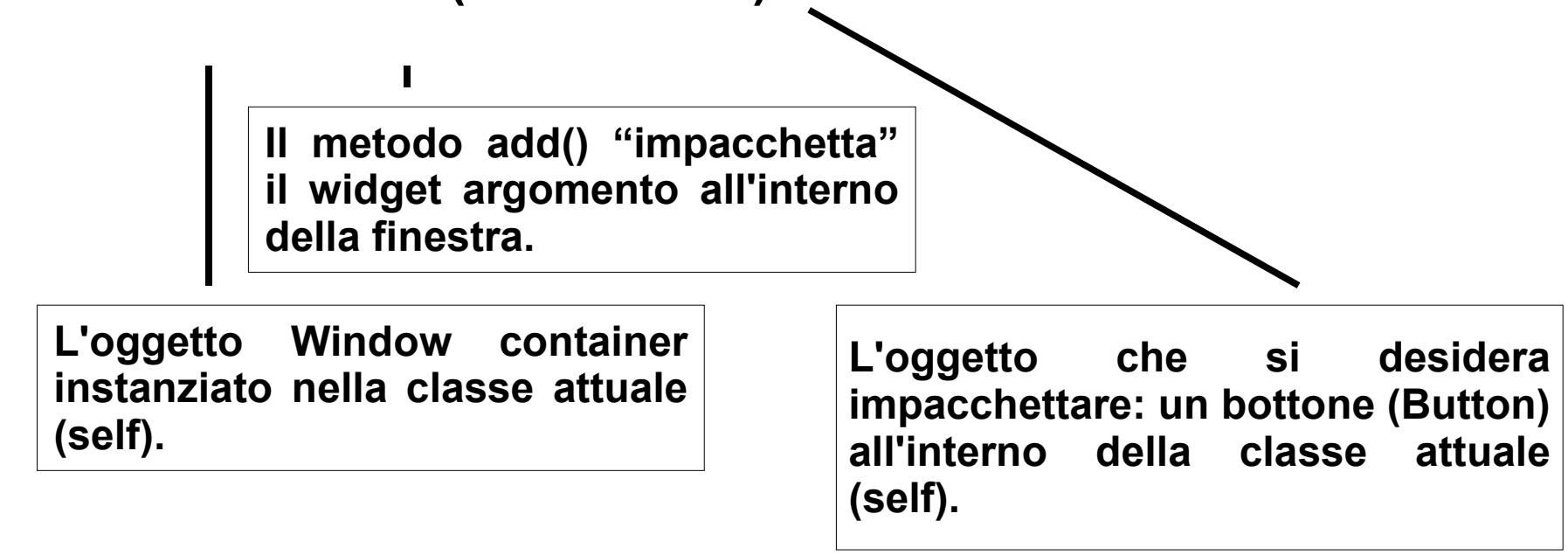
- **Creazione di una finestra contenente un bottone con la scritta “Hello World”**
  - **Illustra il meccanismo di creazione ed impacchettamento dei widget**
- **La pressione del bottone comporta l'uscita dal programma**
  - **Illustra il meccanismo di ricezione di un evento e di invocazione del relativo callback**

Inserimento di un widget all'interno di un widget container

- ESEMPI: helloworld.py

- **Per inserire un widget all'interno di un widget più grande di tipo container si utilizza il metodo add() del widget container**

self.window.add(self.button)



Registrazione di un event handler

- ESEMPI: helloworld.py

- **Per associare (registrare) un gestore di eventi (event handler) ad un evento di un widget si utilizza il metodo connect() del widget**

self.button.connect(“clicked”, self.hello, None)

Callback su altri oggetti

- ESEMPI: helloworld.py

- **Alle volte è comodo eseguire un metodo su un altro widget**
- **Nell'esempio helloworld.py, vogliamo che alla pressione del bottone corrisponde la distruzione della finestra container**
- **Per invocare un metodo di un widget w2 in seguito alla ricezione di un evento su un widget w1 si utilizza il metodo connect_object() del widget w1**

Callback su altri oggetti

- ESEMPI: helloworld.py

self.button.connect_object(“clicked”, gtk.Widget.destroy, self.window)

L'evento che si vuole gestire è “clicked” (pressione del bottone):

Si programma l'esecuzione di un metodo su un altro widget in seguito ad un evento su button.

Il widget su cui sarà invocato il metodo (self.window).

Il metodo che si vuole invocare in seguito alla pressione del bottone (gtk.Widget.destroy()).

Elenco degli eventi

- **Per un elenco esaustivo degli eventi si consulti la seguente pagina Web:**

http://www.pygtk.org/pygtk2tutorial/sec-Events.html

Impacchettamento widget

- ESEMPI: packbox.py

- **Creazione di una finestra contenente una miriade di bottone, organizzati**
  - **orizzontalmente**
  - **verticalmente**
- **Illustra il meccanismo di impacchettamento tramite**
  - **HBox (box orizzontali)**
  - **VBox (box verticali)**

Impacchettamento widget

- ESEMPI: packbox.py

- **Le HBox e VBox sono classi dedicate al raggruppamento di widget in orizzontale ed in verticale, rispettivamente**

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

Parametro "homogeneous": se posto a True, tutti i widget della Box hanno uguale dimensione.

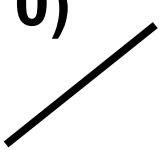
Parametro "spacing": spazio aggiunto fra un widget ed un altro.

Impacchettamento left e right

- ESEMPI: packbox.py

- **L'impacchettamento può avvenire da sinistra verso destra (pack_start()) oppure da destra verso sinistra (pack_end())**

```
hbox.pack_start(child, expand = True, fill = True,
    padding = 0)
hbox.pack_start(child, expand = True, fill = True,
    padding = 0)
```



Parametro "child": il widget da aggiungere alla Box.

Parametro "expand": se posto a True, il widget si espande su uno spazio extra, condiviso con tutti gli altri widget aventi expand=True.

Parametro "fill" (valido se expand = True): se posto a True, lo spazio allocato tramite expand viene addebitato al widget.

Impacchettamento con tabelle

- ESEMPI: table.py

- **L'impacchettamento può avvenire anche attraverso tabelle (gtk.Table)**

```
table = gtk.Table(rows=1, columns=1,
    homogeneous=False)
table.attach(child, left_attach, right_attach, top_attach,
    bottom_attach, xoptions=EXPAND|FILL,
    yoptions=EXPAND|FILL, xpadding=0, ypadding=0)
```

Parametro "homogeneous": se posto a True, gli elementi della tabella hanno tutti la stessa dimensione.

Impacchettamento con tabelle

- ESEMPI: table.py

- **L'impacchettamento può avvenire anche attraverso tabelle (gtk.Table)**

```
table = gtk.Table(rows=1, columns=1,
    homogeneous=False)
table.attach(child, left_attach, right_attach, top_attach,
    bottom_attach, xoptions=EXPAND|FILL,
    yoptions=EXPAND|FILL, xpadding=0, ypadding=0)
```

Metodo attach(): attacca un widget ad un elemento della tabella.

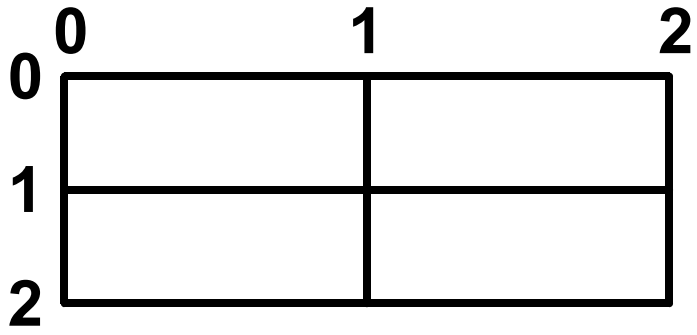
{left,right,top,bottom}_attach: specificano dove piazzare il widget e quanti elementi della tabella usare. Essi rappresentano il numero di colonna (left/right_attach) a cui attaccare il lato sinistro/destro del widget, ed il numero di riga (top/bottom_attach) a cui attaccare il lato superiore/inferiore del widget.

Impacchettamento con tabelle

- ESEMPI: table.py

- **L'impacchettamento può avvenire anche attraverso tabelle (gtk.Table)**

```
table = gtk.Table(rows=1, columns=1,
    homogeneous=False)
table.attach(child, left_attach, right_attach, top_attach,
    bottom_attach, xoptions=EXPAND|FILL,
    yoptions=EXPAND|FILL, xpadding=0, ypadding=0)
```



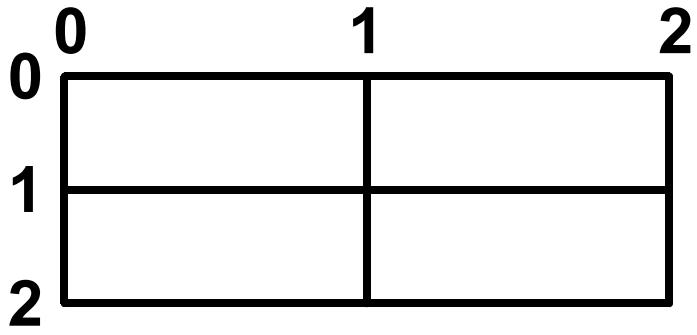
left_attach=1
right_attach=2
top_attach=1
bottom_attach=2

Impacchettamento con tabelle

- ESEMPI: table.py

- **L'impacchettamento può avvenire anche attraverso tabelle (gtk.Table)**

```
table = gtk.Table(rows=1, columns=1,
    homogeneous=False)
table.attach(child, left_attach, right_attach, top_attach,
    bottom_attach, xoptions=EXPAND|FILL,
    yoptions=EXPAND|FILL, xpadding=0, ypadding=0)
```



left_attach=0
right_attach=2
top_attach=0
bottom_attach=1

Impacchettamento con tabelle

- ESEMPI: table.py

- **L'impacchettamento può avvenire anche attraverso tabelle (gtk.Table)**

```
table = gtk.Table(rows=1, columns=1,
    homogeneous=False)
table.attach(child, left_attach, right_attach, top_attach,
    bottom_attach, xoptions=EXPAND|FILL,
    yoptions=EXPAND|FILL, xpadding=0, ypadding=0)
```

Parametro {x,y}options: è una maschera di bit contenente le opzioni sul widget.

Opzioni
FILL: se l'elemento della tabella è più grande del widget, esso si espande sull'intera dimensione dell'elemento.
SHRINK: se la tabella viene rimpicciolita, i widget rimpiccioliscono con essa.
EXPAND: la cella si espande, riempiendo tutto lo spazio offerto dalla tabella.

Alcuni esempi di widget

- Nel resto della lezione viene presentata una carrellata di esempi di widget utili nelle applicazioni
- Iniziamo con i widget relativi a bottoni intervalli (range)
  - Toggle button
  - Check button
  - Radio button
  - Adjustment
  - Scrollbar
  - Scale

Toggle button

- ESEMPI: togglebutton.py

- **Il toggle button è un bottone che rimane in uno stato**
  - **Premuto**
  - **Rilasciato**
- **Esso è implementato dalla classe gtk.ToggleButton()**

toggle_button = gtk.ToggleButton(label=None)

Toggle button

- ESEMPI: togglebutton.py

- **Il metodo get_active() ritorna True se il bottone è premuto**

```
def toggle_button_callback(widget, data):
    if widget.get_active():
        # If control reaches here, the toggle button is down
    else:
        # If control reaches here, the toggle button is up
```

- **Si può forzare la pressione del bottone con il metodo set_active()**

```
toggle_button.set_active(is_active)
```

Check button

- ESEMPI: checkbutton.py

- **Il check button è una alternativa al toggle button**
- **Esso è implementato dalla classe gtk.CheckButton()**

check_button = gtk.CheckButton(label=None)

Radio button

- ESEMPI: radiobutton.py

- **Il radio button permette la pressione di un solo bottone in un gruppo**
- **Esso è implementato dalla classe gtk.RadioButton()**

```
radio_button = gtk.RadioButton(group=None,
    label=None)
```

- **I bottoni sono inseriti associando all'evento toggled un metodo callback**

```
button.connect("toggled", self.callback, "radio button 1")
```

Adjustment

- ESEMPI: rangewidgets.py

- **Il widget Adjustment rappresenta un intervallo di valori impostabile dall'utente**
- **Esso è implementato dalla classe gtk.Adjustment()**

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

- **Parametri**
  - **value: valore iniziale**
  - **lower/upper: limite inferiore/superiore**
  - **step_incr: granularità di un “piccolo” incremento**
  - **page_incr: granularità di un “grande” incremento**
  - **page_size: definisce l'”area visibile” dell'Adjustment**
- **OCCHIO: non è un widget grafico (è di ausilio alle barre grafiche)**

Adjustment

- ESEMPI: rangewidgets.py

- **È possibile reagire al cambiamento di un valore da parte dell'utente Il widget Adjustment associando all'evento value_changed il callback relativo**

```
adj.connect("value_changed", cb_rotate_picture, picture)
```

- **Parametri**
  - **“value_changed”: evento**
  - **cb_rotate_picture: il metodo da invocare**
  - **picture: la variabile contenente il valore aggiornato**

Scrollbar

- ESEMPI: rangewidgets.py

- **La Scrollbar rappresenta una barra di scorrimento**
- **Essa è implementata dalla classi gtk.HSscrollbar() e gtk.VSscrollbar()**

```
hscrollbar = gtk.HSscrollbar(adjustment=None)
vscrollbar = gtk.VSscrollbar(adjustment=None)
```

- **Alcuni metodi utili**
  - **scrollbar.set_update_policy(policy): imposta la modalità di aggiornamento dei valori**
    - **CONTINUOUS (default): mentre la scrollbar viene mossa, i valori si aggiornano continuamente**
    - **DISCONTINUOUS: il valore si aggiorna solo al termine del movimento della scrollbar**

Scale

- ESEMPI: rangewidgets.py

- **La Scale è una barra utilizzata per modificare graficamente un valore in un intervallo**
- **Essa è implementata dalle classi gtk.HScale() e gtk.VScale()**

```
vscale = gtk.VScale(adjustment=None)
hscale = gtk.HScale(adjustment=None)
```

- **Alcuni metodi utili**
  - **scale.set_value_pos(pos): imposta un valore sulla scala**
  - **scale.set_digits(digits): imposta il numero di cifre dopo la virgola**

Alcuni esempi di widget

- Continuiamo la carrellata con alcuni widget di contorno
  - Label
  - Barre di progresso
  - Immagini
  - Barre di stato
  - Caselle di testo
  - Selettori di file
  - Menu

Label

- ESEMPI: label.py

- **La Label è un widget contenente testo non modificabile dall'utente**
- **Essa è implementata dalla classe gtk.Label()**

label = gtk.Label(str)

- **Alcuni metodi utili**
  - **label.set_text(str): imposta il testo della label**
  - **label.set_text(str): ritorna il testo della label**
  - **label.set_justify(jtype): imposta la formattazione del testo della label**
    - **JUSTIFY_LEFT (default): formattato a sinistra**
    - **JUSTIFY_RIGHT: formattato a destra**
    - **JUSTIFY_CENTER: formattato centrale**
    - **JUSTIFY_FILL: giustificato**

Progress bar

- ESEMPI: progressbar.py

- **La Progress bar è un widget rappresentante una barra di progresso**
- **Essa è implementata dalla classe gtk.ProgressBar()**

```
progressbar = gtk.ProgressBar(adjustment=None)
```

- **Alcuni metodi utili**
  - **progressbar.set_fraction(fraction): imposta la quantità di lavoro compiuto (e mostrato dalla barra di progresso); fraction è un valore fra 0 ed 1**

Progress bar

- ESEMPI: progressbar.py

- **La Progress bar è un widget rappresentante una barra di progresso**
- **Essa è implementata dalla classe gtk.ProgressBar()**

```
progressbar = gtk.ProgressBar(adjustment=None)
```

- **Alcuni metodi utili**
  - **progressbar.set_orientation(orientation): imposta l'orientamento della barra di progresso**
    - **PROGRESS_LEFT_TO_RIGHT**
    - **PROGRESS_RIGHT_TO_LEFT**
    - **PROGRESS_BOTTOM_TO_TOP**
    - **PROGRESS_TOP_TO_BOTTOM**

Progress bar

- ESEMPI: progressbar.py

- **La Progress bar è un widget rappresentante una barra di progresso**
- **Essa è implementata dalla classe gtk.ProgressBar()**

```
progressbar = gtk.ProgressBar(adjustment=None)
```

- **Alcuni metodi utili**
  - **progressbar.pulse(): fa muovere la barra a sinistra e a destra ripetutamente**
  - **progressbar.set_pulse_step(fraction): imposta la variazione di spostamento della barra in caso di pulse()**
  - **progressbar.set_text(text): imposta il testo della barra di progresso**

Immagini

- ESEMPI: images.py

- Il widget Image rappresenta una immagine
- Essa è implementata dalla classe gtk.Image()

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

- Una immagine può essere creata
  - da strutture dati preesistenti e rappresentanti un disegno (Pixbuf, Pixmap)
  - da un file preesistente
- Alcuni metodi utili
  - image.set_from_file(filename): carica un file, lo traduce nella struttura dati corretta e lo importa

Barra di stato

- ESEMPI: statusbar.py

- **Il widget StatusBar rappresenta una barra di stato**
- **Essa è implementata dalla classe gtk.StatusBar()**

```
statusbar = gtk.Statusbar()
```

- **I messaggi scritti nella StatusBar sono organizzati secondo uno stack**
  - **Pop dallo stack: viene visualizzato il messaggio precedente**
  - **Push sullo stack: viene accodato un messaggio per la visione**

Barra di stato

- ESEMPI: statusbar.py

- **Il widget StatusBar rappresenta una barra di stato**
- **Essa è implementata dalla classe gtk.StatusBar()**

```
statusbar = gtk.Statusbar()
```

- **Diverse parti dell'applicazione condividono la StatusBar**
  - **A ciascun "utente" della StatusBar è associato un context identifier che lo identifica univocamente**
  - **Ciascun messaggio è accompagnato dal context identifier opportuno**
  - **Lo stack dei messaggi è uno solo**

Barra di stato

- ESEMPI: statusbar.py

- **Il widget StatusBar rappresenta una barra di stato**
- **Essa è implementata dalla classe gtk.StatusBar()**

statusbar = gtk.Statusbar()

- **Alcuni metodi utili**
  - **context_id = statusbar.get_context_id(context_description): recupera il context identifier**
  - **message_id = statusbar.push(context_id, text): spinge nello stack il messaggio text per conto di context_id ed ottiene un message_id che può essere usato per la rimozione futura**

Barra di stato

- ESEMPI: statusbar.py

- **Il widget StatusBar rappresenta una barra di stato**
- **Essa è implementata dalla classe gtk.StatusBar()**

```
statusbar = gtk.Statusbar()
```

- **Alcuni metodi utili**
  - **statusbar.pop(context_id): toglie dallo stack il messaggio più in alto appartenente a context_id**
  - **statusbar.remove(context_id, message_id): toglie dallo stack il messaggio message_id appartenente a context_id**

Casella di testo

- ESEMPI: entry.py

- **Il widget Entry rappresenta una casella di testo modificabile dall'utente**
- **Essa è implementata dalla classe gtk.Entry()**
- **entry = gtk.Entry(max=0)**
  - **Il parametro max rappresenta la lunghezza massima della casella di testo**
- **Alcuni metodi utili**
  - **entry.set_max_length(max): reimposta la lunghezza massima della casella di testo**
  - **entry.set_text(text): imposta il messaggio della casella di testo**
  - **entry.insert_text(text, position=0): inserisce il testo text nella posizione position**

Casella di testo

- ESEMPI: entry.py

- **Il widget Entry rappresenta una casella di testo modificabile dall'utente**
- **Essa è implementata dalla classe gtk.Entry()**
- **entry = gtk.Entry(max=0)**
  - **Il parametro max rappresenta la lunghezza massima della casella di testo**
- **Alcuni metodi utili**
  - **text = entry.get_text(): recupera il testo**
  - **entry.set_editable(is_editable): imposta la casella di testo in modalità "editabile" dall'utente**
  - **entry.set_visibility(visible): rende visibile il testo della casella (immissione testo in chiaro vs. password)**
  - **entry.select_region(start, end): seleziona la regione di testo dalla posizione start alla posizione end**

File selector

- ESEMPI: images.py

- **Il widget FileSelection implementa un dialogo di selezione dei file**
- **Esso è implementato dalla classe gtk.FileSelection()**
- **filesel = gtk.FileSelection(title=None)**
- **Alcuni metodi utili**
  - **filesel.set_filename(filename): imposta il nome del file**
  - **filename = filesel.get_filename(): recupera il nome del file**

Menu

- ESEMPI: menu.py

- **Il widget Menu è implementato dalla classe gtk.MenuBar()**

menu_bar = gtk.MenuBar()

- **Il metodo Menu() inizializza un nuovo menu di primo livello**

menu = gtk.Menu()

- **Il metodo MenuItem() genera voci di menu**

menu_item = gtk.MenuItem(label=None)

Menu

- ESEMPI: menu.py

- **Il metodo append() della classe Menu aggancia le voci di menu al menu opportuno**

file_menu.append(open_item)

- **Per aggiungere dei submenu, si usa il metodo submenu() della classe MenuItem**

file_item.set_submenu(file_menu)

Implementazione semplificata: ItemFactory

- ESEMPI: itemfactory.py

- **Il menu può essere implementato in maniera semplificata tramite la classe ItemFactory**
- **Si guardi l'esempio ItemFactory.py**