Full Circle

LA RIVISTA INDIPENDENTE PER LA COMUNITÀ LINUX UBUNTU

EDIZIONE SPECIALE SERIE PROGRAMMAZIONE

EDIZIONE SPECIALE
SERIE PROGRAMMAZIONE

python™

PROGRAMMARE IN PYTHON
VOLUME 5

Cos'è Full Circle

Full Circle è una rivista gratuita e indipendente, dedicata alla famiglia Ubuntu dei sistemi operativi Linux. Ogni mese pubblica utili articoli tecnici e articoli inviati dai lettori.

Full Circle ha anche un podcast di supporto, il Full Circle Podcast, con gli stessi argomenti della rivista e altre interessanti notizie.

Si prega di notare che questa edizione speciale viene fornita senza alcuna garanzia: né chi ha contribuito né la rivista Full Circle hanno alcuna responsabilità circa perdite di dati o danni che possano derivare ai computer o alle apparecchiature dei lettori dall'applicazione di quanto pubblicato.

Full Circle

LA RIVISTA INDIPENDENTE PER LA COMUNITÀ LINUX UBUNTU

Ecco a voi un altro 'Speciale monotematico'

Come richiesto dai nostri lettori, stiamo assemblando in edizioni dedicate alcuni degli articoli pubblicati in serie.

Quella che avete davanti è la ristampa della serie '**Programmare in Python' parti 27-31,** pubblicata nei numeri 53-59: e già, l'impareggiabile professor Gregg Walters si è concesso una pausa durante questo periodo.

Vi preghiamo di tenere conto della data di pubblicazione: le versioni attuali di hardware e software potrebbero essere diverse rispetto ad allora. Controllate il vostro hardware e il vostro software prima di provare quanto descritto nelle guide di queste edizioni speciali. Potreste avere versioni più recenti del software installato o disponibile nei repository delle vostre distribuzioni.

Buon divertimento!

Come contattarci

Sito web:
http://www.fullcirclemagazine.org/

Forum:
http://ubuntuforums.org/forumdisplay.php?f=270

IRC:#fullcirclemagazine su chat.freenode.net

Gruppo editoriale

Capo redattore: Ronnie Tucker (aka: RonnieTucker)
ronnie@fullcirclemagazine.org

Webmaster: Rob Kerfia (aka: admin / linuxgeekery-admin@fullcirclemagazine.org

Modifiche e Correzioni
Mike Kennedy, Lucas Westermann, Gord Campbell, Robert Orsino, Josh Hertel, Bert Jerred

Si ringrazia la Canonical e i tanti gruppi di traduzione nel mondo.

Programmare in Python - Parte 27

Scritto da Greg Walters

Se avete mai aspettato in fila per comprare un biglietto del cinema, siete stati in coda. Se avete mai aspettato nel traffico all'ora di punta, siete stati in coda. Se avete mai aspettato in un ufficio pubblico con in mano uno di quei bigliettini che dicevano che eravate il numero 98 e il tabellone indicava "serviamo il numero 42", siete stati in coda.

Nel mondo dei computer le code sono comuni. Come utente, la maggior parte delle volte, non è necessario preoccuparsene. Sono invisibili. Ma se dovete avere a che fare con eventi in tempo reale allora è necessario conoscerle. Si tratta semplicemente di un dato di un qualche tipo, che attende di essere processato. Una volta entrato nella coda, vi resta finché non viene richiesto e quindi viene rimosso. Non è possibile recuperare il valore del dato successivo finché non lo si estrae dalla coda. Non potete, per esempio, recuperare il valore del 15° elemento della coda. Dovete prima accedere ai restanti 14 elementi. Una volta che l'accesso è possibile, esce dalla coda. È andato, e a meno che non venga salvato in una variabile a lungo termine, non è più possibile recuperarlo.

Ci sono molteplici tipi di code. Le più comuni sono FIFO (First In, First Out)(primo a entrare, primo a uscire), LIFO (Last In, First Out)(ultimo ad entrare, primo ad uscire), a Priorità e ad Anello. Parleremo delle code ad anello la prossima volta.

Le code FIFO sono quelle che osserviamo nella vita di tutti i giorni. Tutti gli esempi fatti prima sono code FIFO. La prima persona nella fila è servita per prima, si sposta, quindi ognuno si muove di un posto. In un buffer FIFO non c'è nessun limite al numero di elementi che può contenere, entro limiti ragionevoli. Vengono impilati in ordine. Quando un elemento è processato, viene estratto dalla coda e il resto si muove verso l'uscita di una posizione.

Le code LIFO sono meno comuni, ma si possono comunque fare esempi presi dal mondo reale. Quello che mi viene in mente più facilmente è una pila di piatti nell'armadio della cucina. Quando i piatti sono lavati e asciugati, vengono riposti nell'armadio. L'ultimo della fila è il primo che viene preso per essere usato. Tutti gli altri devono aspettare, forse per giorni, prima di essere utilizzati. È una cosa buona che la coda per il biglietto del cinema sia di tipo FIFO, vero? Come la coda FIFO, entro limiti ragionevoli, non c'è nessun limite alla dimensione di una coda LIFO. Il primo elemento nella coda deve aspettare man mano che nuovi elementi sono estratti dal buffer (piatti estratti dalla pila) finché risulta l'ultimo rimasto.

Le code a priorità sono per molte persone un po' più difficili da immaginare correttamente fuori dal contesto. Pensate ad una compagnia che ha una sola stampante. Tutti usano quell'unica stampante. I lavori di stampa sono gestiti dalla priorità del dipartimento. L'ufficio paghe ha una priorità superiore (e meno male) di, diciamo, te, un programmatore. Voi avete una priorità superiore (e meno male) rispetto all'addetto all'accettazione. Quindi, in breve, i dati che hanno una priorità superiore sono gestiti, ed escono fuori dalla coda, prima di quelli che hanno una priorità inferiore.

> **Ci sono molteplici tipi di code. Le più comuni sono FIFO (First In, First Out)(primo a entrare, primo a uscire), LIFO (Last In, First Out)(ultimo ad entrare, primo ad uscire), a Priorità e ad Anello.**

FIFO

Le code FIFO sono semplici da visualizzare in termini di dati. Una lista python è una semplice rappresentazione mentale. Considerate questa lista...

```
[1,2,3,4,5,6,7,8,9,10]
```

Ci sono 10 elementi nella lista. In quanto tale, potete accedere ai suoi elementi tramite un indice. Invece, in una coda non è possibile accedere agli elementi attraverso un indice. Dovete gestire il prossimo elemento nella coda e la lista non è statica. È MOLTO dinamica. Quando noi richiediamo il prossimo elemento nella coda, questo viene rimosso. Così usando l'esempio sopra, richiedete un

```
import Queue
fifo = Queue.Queue()
for i in range(5):
    fifo.put(i)

while not fifo.empty():
    print fifo.get()
```

elemento dalla coda. Viene restituito il primo elemento (1) e la coda quindi assomiglia a questa.

```
[2,3,4,5,6,7,8,9,10]
```

Richiedetene altri due e ottenete 2, quindi 3 e quindi la coda assomiglia a questa.

```
[4,5,6,7,8,9,10]
```

Sono sicuro vi siete fatti un'idea. Python fornisce una semplice libreria, sorprendentemente sufficiente, chiamata Queue, che funziona bene per code di piccola e media dimensione, fino a 500 elementi. In alto c'è un semplice esempio che la mostra.

In questo esempio, inizializziamo la coda (fifo = Queue.Queue()) quindi vi inseriamo i numeri da 0 a 4 (fifo.put(i)). Quindi usiamo il metodo interno .get() per estrarre gli elementi dalla coda finché non è vuota, .empty(). Quello che viene restituito è

```
import Queue

fifo = Queue.Queue(12)
for i in range(13):
    if not fifo.full():
        fifo.put(i)

while not fifo.empty():
    print fifo.get()
```

0,1,2,3,4. Potete anche definire il numero massimo di elementi che la coda deve contenere inizializzandola con la dimensione, come in questo esempio.

```
fifo = Queue.Queue(300)
```

Una volta che il numero massimo di elementi è stato caricato, Queue blocca ogni inserimento aggiuntivo. Questo ha come sgradevole effetto quello di far sembrare il programma come "bloccato". Il modo più semplice per aggirare il problema è usare il controllo Queue.full() (in alto a destra).

In questo caso, la coda è impostata per un massimo di 12 elementi. Quanto aggiungiamo elementi, iniziamo con 0 e arriviamo a 11. Quando inseriamo il numero 12, però, il buffer è già pieno. Poiché verifichiamo se il buffer è pieno prima di inserire un elemento, l'ultimo viene semplicemente scartato.

Ci sono altre opzioni, ma possono causare altri effetti secondari, e ce ne occuperemo in un articolo futuro. Quindi, nella maggior parte delle occasioni, assicuratevi di usare una coda senza limiti o di avere comunque abbastanza spazio per i vostri bisogni.

LIFO

```
import Queue
lifo = Queue.LifoQueue()
for i in range(5):
    lifo.put(i)
while not lifo.empty():
    print lifo.get()
```

La libreria Queue supporta anche le code LIFO. Useremo la lista precedente come esempio visuale. Impostiamo la nostra coda, appare come questa:

```
[1,2,3,4,5,6,7,8,9,10]
```

estraiamo tre elementi dalla coda, che appare come questa:

```
[1,2,3,4,5,6,7]
```

Ricordate che in una coda LIFO, gli elementi sono rimossi nell'ordine ULTIMO a entrare PRIMO a uscire.

Ecco il semplice esempio modificato per la coda LIFO:

```
pq = Queue.PriorityQueue()
pq.put((3,'Medium 1'))
pq.put((4,'Medium 2'))
pq.put((10,'Low'))
pq.put((1,'high'))

while not pq.empty():
    nex = pq.get()
    print nex
    print nex[1]
```

Quando lo eseguiamo, otteniamo "4,3,2,1,0".

Come per la coda FIFO, avete la possibilità di impostarne la dimensione, e potete usare il controllo .full().

Priorità

Anche se non è usata spesso, una

```
(1, 'high')
high
(3, 'Medium')
Medium
(4, 'Medium')
Medium
(10, 'Low')
Low
```

coda a priorità può a volte essere utile. È molto simile alle precedenti, ma dobbiamo passarle una tupla che contiene sia la priorità che il dato. Ecco un esempio che utilizza la libreria Queue:

Prima, inizializziamo la coda. Quindi inseriamo quattro elementi. Notate come venga usato il formato (priorità, dato) per inserire i dati. La libreria ordina i nostri dati in maniera ascendente in base al valore di priorità. Quando estraiamo il dato, questo è in forma di tupla, proprio come l'abbiamo inserito. È possibile usare un indice come indirizzo del dato. Quello che otteniamo è...

Nei primi due esempi, abbiamo semplicemente stampato i dati che venivano fuori dalla coda. Questo va bene per questi esempi, ma nella programmazione nel mondo reale, probabilmente è necessario fare qualcosa con l'informazione appena estratta dalla coda, altrimenti è perduta. Quando usiamo 'print fifo.get', il dato è inviato al terminale pere poi essere distrutto. Giusto una cosa da tenere a mente. Ora usiamo qualcosa di quello che abbiamo già imparato su tkinter per creare un programma dimostrativo sulle code. Conterrà due cornici. La prima conterrà (per l'utente) tre pulsanti. Uno per la coda FIFO, uno per la coda LIFO ed uno per la coda a priorità. La seconda cornice conterrà un widget di inserimento, due pulsanti, uno per aggiungere alla coda e uno per estrarre dalla coda, e tre etichette, una che mostra quando la coda è vuota, una che mostra quando la coda è piena, e una che mostra cosa è stato estratto. Scriveremo anche del codice per centrare automaticamente la finestra sullo schermo. In alto sinistra c'è l'inizio del nostro codice.

```
import sys
from Tkinter import *
import ttk
import tkMessageBox
import Queue

class QueueTest:
    def __init__(self,master = None):
        self.DefineVars()
        f = self.BuildWidgets(master)
        self.PlaceWidgets(f)
        self.ShowStatus()
```

```
    def DefineVars(self):
        self.QueueType = ''
        self.FullStatus = StringVar()
        self.EmptyStatus = StringVar()
        self.Item = StringVar()
        self.Output = StringVar()
        # Define the queues
        self.fifo = Queue.Queue(10)
        self.lifo = Queue.LifoQueue(10)
        self.pq = Queue.PriorityQueue(10)
        self.obj = self.fifo
```

Qui abbiamo gli import e l'inizio della classe. Come prima, creiamo la funzione __init__ con le routine DefineVars, BuildWidgets e PlaceWidgets. Ne abbiamo anche una chiamata ShowStatus (in alto a destra) che... bé, mostra lo stato della coda.

Ora creiamo la funzione

```
    def BuildWidgets(self,master):
        # Define our widgets
        frame = Frame(master)
        self.f1 = Frame(frame,
            relief = SUNKEN,
            borderwidth=2,
            width = 300,
            padx = 3,
            pady = 3
        )
        self.btnFifo = Button(self.f1,
            text = "FIFO"
        )
        self.btnFifo.bind('<ButtonRelease-1>',
            lambda e: self.btnMain(1)
        )
        self.btnLifo = Button(self.f1,
            text = "LIFO"
        )
        self.btnLifo.bind('<ButtonRelease-1>',
            lambda e: self.btnMain(2)
        )
        self.btnPriority = Button(self.f1,
            text = "PRIORITY"
        )
        self.btnPriority.bind('<ButtonRelease-1>',
            lambda e: self.btnMain(3)
        )
```

DefineVars. Abbiamo quattro oggetti StringVar(), una variabile vuota chiamata QueueType e tre oggetti coda, uno per ciascun tipo di coda con cui giocheremo. Per gli scopi di questa demo abbiamo impostato la dimensione massima delle code a 10. Abbiamo anche creato un oggetto chiamato obj e lo abbiamo assegnato alla coda FIFO. Quando selezioniamo un tipo di coda attraverso i pulsanti, assegnamo quest'oggetto alla coda che vogliamo. In questo modo, la coda è mantenuta anche quando si passa ad un altro tipo (il codice è nella pagina precedente, in basso a destra).

Qui iniziamo la definizione dei widget. Creiamo la nostra prima cornice, i tre pulsanti e le loro connessioni. Notate che stiamo usando la stessa routine per gestire le connessioni di supporto. Ciascun pulsante invia un valore alla funzione di callback che denota quale pulsante è stato premuto. Avremmo potuto facilmente creare una routine dedicata per ciascun pulsante. Comunque, dato che i tre pulsanti si occuperanno di un compito comune, ho pensato fosse meglio lavorarci in gruppo (il codice mostrato a destra).

A seguire (in basso a destra), impostiamo la seconda cornice, il widget di inserimento e i due pulsanti. L'unica cosa qui che è fuori dall'ordinario è la connessione per il widget di inserimento. Colleghiamo la funzione self.AddToQueue al tasto Invio. In questo modo l'utente non deve usare il mouse per aggiungere il dato. Basta inserirlo nel widget, se vuole.

Qui (pagina seguente, in basso) c'è la definizione degli ultimi tre widget. Tutti e tre sono etichette. Impostiamo l'attributo textvariable alle variabili definite precedentemente. Se ricordate, quando quella variabile cambia, lo fa anche l'etichetta. Facciamo qualcosa un po' differente nell'etichetta lblData. Useremo un carattere diverso per evidenziarla quando mostriamo il dato estratto dalla coda. Ricordate che dobbiamo restituire la cornice oggetto così che possa essere usata nella funzione PlaceWidget.

Questo (pagina seguente, al centro) è l'inizio della funzione PlaceWidget. Notate che inseriamo cinque etichette vuote proprio all'inizio della finestra radice. L'ho fatto per inserire dello spazio. È un metodo semplice per "barare" e rendere la disposizione della finestra più semplice. Quindi impostiamo la

```
self.f2 = Frame(frame,
    relief = SUNKEN,
    borderwidth=2,
    width = 300,
    padx = 3,
    pady = 3
)
self.txtAdd = Entry(self.f2,
    width=5,
    textvar=self.Item
)
self.txtAdd.bind('<Return>',self.AddToQueue)
self.btnAdd = Button(self.f2,
    text='Add to Queue',
    padx = 3,
    pady = 3
)
self.btnAdd.bind('<ButtonRelease-1>',self.AddToQueue)
self.btnGet = Button(self.f2,
    text='Get Next Item',
    padx = 3,
    pady = 3
)
self.btnGet.bind('<ButtonRelease-1>',self.GetFromQueue)
```

```
        self.lblEmpty = Label(self.f2,
            textvariable=self.EmptyStatus,
            relief=FLAT
        )
        self.lblFull = Label(self.f2,
            textvariable=self.FullStatus,
            relief=FLAT
        )
        self.lblData = Label(self.f2,
            textvariable=self.Output,
            relief = FLAT,
            font=("Helvetica", 16),
            padx = 5
        )

        return frame
```

prima cornice, quindi un'altra etichetta "truffaldina", quindi tre pulsanti.

Qui posizioniamo la seconda cornice, un'altra etichetta 'truffaldina', e i rimanenti widget.

```
def Quit(self):
    sys.exit()
```

A seguire abbiamo la consueta funzione quit che chiama semplicemente sys.exit() (in alto a destra).

Ora la funzione di supporto del pulsante principale, btnMain. Ricordate che stiamo inviando (attraverso il parametro p1) l'informazione relativa al pulsante premuto. Usiamo la variabile self.QueueType come riferimento al tipo di coda su cui stiamo lavorando, quindi assegniamo self.obj alla coda appropriata e per finire cambiamo il titolo della finestra per mostrare il tipo di coda che stiamo usando. Dopo questo, stampiamo il tipo di coda nella finestra di terminale (non è realmente necessario farlo) e chiamiamo la routine ShowStatus. A seguire (pagina seguente in alto a destra) creeremo la funzione ShowStatus.

Come potete vedere, è

```
def btnMain(self,p1):
    if p1 == 1:
        self.QueueType = 'FIFO'
        self.obj = self.fifo
        root.title('Queue Tests - FIFO')
    elif p1 == 2:
        self.QueueType = 'LIFO'
        self.obj = self.lifo
        root.title('Queue Tests - LIFO')
    elif p1 == 3:
        self.QueueType = 'PRIORITY'
        self.obj = self.pq
        root.title('Queue Tests - Priority')

    print self.QueueType
    self.ShowStatus()
```

```
self.f2.grid(column = 0,row = 2,sticky='nsew',columnspan=5,padx = 5, pady = 5)
l = Label(self.f2,text='',width = 15,anchor = 'e').grid(column = 0, row = 0)
self.txtAdd.grid(column=1,row=0)
self.btnAdd.grid(column=2,row=0)
self.btnGet.grid(column=3,row=0)
self.lblEmpty.grid(column=2,row=1)
self.lblFull.grid(column=3,row = 1)
self.lblData.grid(column = 4,row = 0)
```

```
def PlaceWidgets(self, master):
    frame = master
    # Place the widgets
    frame.grid(column = 0, row = 0)
    l = Label(frame,text='',relief=FLAT,width = 15, anchor = 'e').grid(column = 0, row = 0)
    l = Label(frame,text='',relief=FLAT,width = 15, anchor = 'e').grid(column = 1, row = 0)
    l = Label(frame,text='',relief=FLAT,width = 15, anchor = 'e').grid(column = 2, row = 0)
    l = Label(frame,text='',relief=FLAT,width = 15, anchor = 'e').grid(column = 3, row = 0)
    l = Label(frame,text='',relief=FLAT,width = 15, anchor = 'e').grid(column = 4, row = 0)

    self.f1.grid(column = 0,row = 1,sticky='nsew',columnspan=5,padx = 5,pady = 5)
    l = Label(self.f1,text='',width = 25,anchor = 'e').grid(column = 0, row = 0)
    self.btnFifo.grid(column = 1,row = 0,padx = 4)
    self.btnLifo.grid(column = 2,row = 0,padx = 4)
    self.btnPriority.grid(column = 3, row = 0, padx = 4)
```

abbastanza semplice. Impostiamo lo stato appropriato delle etichette così che mostrino se la coda che stiamo usando è piena, vuota o qualcos'altro nell'intervallo.

La funzione AddToQueue (pagina seguente, in basso a destra) è anch'essa semplice. Recuperiamo il dato dal box di inserimento usando la funzione .get(). Quindi controlliamo per vedere se l'attuale tipo di coda è di tipo priorità. In caso affermativo, dobbiamo essere sicuri che sia nel formato corretto. Lo facciamo controllando se c'è la virgola. Se non c'è, avvisiamo l'utente attraverso una messaggio di errore. Se tutto sembra corretto, controlliamo per vedere se la coda attuale è piena. Ricordate, se la coda è piena la funzione di inserimento si blocca e il programma non risponderà. Se tutto è a posto, aggiungiamo l'elemento alla coda e aggiorniamo il suo stato.

La funzione GetFromQueue (al centro a destra) è ancora più semplice. Controlliamo per vedere se la coda è vuota così da non incorrere in un problema di blocco, e, se non lo è, estraiamo il dato, lo mostriamo e aggiorniamo lo stato.

Stiamo arrivando alla fine della nostra applicazione. Ecco la funzione per centrare la finestra (in alto a sinistra). Prima recuperiamo la larghezza e l'altezza dello schermo. Quindi recuperiamo la larghezza e l'altezza della finestra usando le funzioni winfo_reqwidth() e winfo_reqheight() definite in tkinter. Queste funzioni, quando chiamate al momento giusto, restituiscono la larghezza e l'altezza della finestra in base al posizionamento dei widget. Se la chiamiamo troppo presto, otteniamo i dati, ma non sono

```
if __name__ == '__main__':
    def Center(window):
        # Get the width and height of the screen
        sw = window.winfo_screenwidth()
        sh = window.winfo_screenheight()
        # Get the width and height of the window
        rw = window.winfo_reqwidth()
        rh = window.winfo_reqheight()
        xc = (sw-rw)/2
        yc = (sh-rh)/2
        window.geometry("%dx%d+%d+%d"%(rw,rh,xc,yc))
        window.deiconify()
```

```
def ShowStatus(self):
    # Check for Empty
    if self.obj.empty() == True:
        self.EmptyStatus.set('Empty')
    else:
        self.EmptyStatus.set('')
    # Check for Full
    if self.obj.full() == True:
        self.FullStatus.set('FULL')
    else:
        self.FullStatus.set('')
```

```
def GetFromQueue(self,p1):
    self.Output.set('')
    if not self.obj.empty():
        temp = self.obj.get()
        self.Output.set("Pulled
{0}".format(temp))
    self.ShowStatus()
```

```
def AddToQueue(self,p1):
    temp = self.Item.get()
    if self.QueueType == 'PRIORITY':
        commapos = temp.find(',')
        if commapos == -1:
            print "ERROR"
            tkMessageBox.showerror('Queue Demo',
                'Priority entry must be in format\r(priority,data)')
        else:
            self.obj.put(self.Item.get())
    elif not self.obj.full():
        self.obj.put(self.Item.get())
    self.Item.set('')
    self.ShowStatus()
```

quelli di cui abbiamo veramente bisogno. Quindi sottraiamo la larghezza della finestra dalla larghezza dello schermo e dividiamo per due, e facciamo la stessa cosa per l'altezza. Quindi usiamo questi dati per chiamare geometry. Nella maggior parte delle istanze, questo funziona alla perfezione. Però, potrebbero esserci dei casi in cui sia necessario inserire a mano larghezza e altezza.

```
root = Tk()
root.title('Queue Tests - FIFO')
demo = QueueTest(root)
root.after(3,Center,root)
root.mainloop()
```

Per finire, instanziamo la finestra radice, impostiamo il titolo di base, instanziamo la classe QueueTest. Quindi chiamiamo root.after, che attende X numero di millisecondi (in questo caso tre) dopo che la finestra è stata instanziata e quindi chiama la funzione center. In questo modo la finestra è stata completamente impostata ed è pronta, così che è possibile recuperare la larghezza e l'altezza. Potrebbe essere necessario aggiustare un po' il tempo di attesa. Alcuni computer sono più veloci di altri. Il valore '3' funziona bene sulla mia macchina, nel vostro caso potrebbe variare. Alla fine ma non per importanza, chiamiamo mainloop della finestra per eseguire l'applicazione.

Giocando con le code noterete che mettendo un dato in una coda (diciamo la coda FIFO) e quindi passando ad un altro tipo (diciamo LIFO), il dato inserito nella prima coda è ancora lì e ci aspetta. Potete completamente o parzialmente riempire tutte e tre le code, quindi iniziare a giocare con esse. Bene, questo è tutto per questa volta.

Divertitevi con le vostre code. Il codice può essere trovato all'indirizzo http://pastebin.com/5BBUiDce.

Greg Walters è il proprietario della RainyDay Solutions, LLC, una società di consulenza in Colorado e programma dal 1972. Ama cucinare, fare escursioni, ascoltare musica e passare il tempo con la sua famiglia. Il suo sito web è www.thedesignatedgeek.com.

Programmare in Python - Parte 28

Scritto da Greg Walters

Ci accingiamo a esplorare altri widget messi a disposizione da tkinter. Questa volta ci occuperemo di menù, combo box, spin box, barra separatrice, barra di progresso e notebook. Parliamone uno alla volta.

Avete di certo visto i menù praticamente in ogni applicazione che avete usato. Tkinter rende la loro procedura di creazione MOLTO semplice. I combo box sono simili al widget lista visto nell'ultimo articolo, differenziandosene per il fatto che i suoi elementi "compaiono" invece di essere sempre visibili. Il widget spin box sono utili per fissare un arco di valori tra cui scorrere. Per esempio, se si vuole che l'utente scelga un valore intero compreso tra 1 e 100 possiamo usare un semplice spin box. Le barre di progresso rappresentano un metodo utilissimo per mostrare che la vostra applicazione non si è bloccata nell'esecuzione di una procedura lunga, come leggere i record di un database. Può mostrare la percentuale di completamento dell'azione. Esistono due tipi di questo widget, determinato e indeterminato. Si usa il tipo determinato quando si conosce a priori il numero di elementi da elaborare. Se questo valore è ignoto o se in un dato momento è impossibile conoscere la percentuale di completamento, allora userete la versione indeterminata. Le useremo entrambe. Per finire, il widget notebook (o widget a schede) è usato molte volte nelle schermate di configurazione. È possibile raggruppare logicamente una serie di widget in ciascuna scheda.

Allora, iniziamo. Come al solito, creeremo un'applicazione base e la popoleremo via via con i vari widget. A destra trovate la prima parte della nostra applicazione. Per lo più la conoscete già.

Salvate il tutto come widgetdemo2a.py. Ricordate che lo useremo come base per creare il demo completo. Iniziamo a creare il menù. Ecco i passi necessari. Prima definiamo una variabile per contenere l'istanza del menù. Come per gli altri widget utilizzati, il formato è...

```
OurVariable = Widget(parent, options).
```

In questo caso stiamo usando il

```
import sys
from Tkinter import *
import ttk
# Shows how to create a menu
class WidgetDemo2:

    def __init__(self,master = None):
        self.DefineVars()
        f = self.BuildWidgets(master)
        self.PlaceWidgets(f)

    def DefineVars(self):
        pass
```

E qui ecco la parte inferiore del programma. Ancora, lo avete visto precedentemente. Niente di nuovo.

```
if __name__ == '__main__':
    def Center(window):
        # Get the width and height of the screen
        sw = window.winfo_screenwidth()
        sh = window.winfo_screenheight()
        # Get the width and height of the window
        rw = window.winfo_reqwidth()
        rh = window.winfo_reqheight()
        xc = (sw-rw)/2
        yc = (sh-rh)/2
        print "{0}x{1}".format(rw,rh)
        window.geometry("%dx%d+%d+%d"%(rw,rh,xc,yc))
        window.deiconify()

    root = Tk()
    root.title('More Widgets Demo')
    demo = WidgetDemo2(root)
    root.after(13,Center,root)
    root.mainloop()
```

widget Menu assegnandolo alla finestra principale genitore. Lo facciamo nella funzione BuildWidgets. Quindi creiamo un altro menù, questa volta chiamato filemenu. Aggiungiamo comandi e separatori quando necessario. Finiamo aggiungendo filemenu alla barra dei menù e ripetiamo la procedura. Nel nostro esempio abbiamo menubar, i menù File, Edit e Help (in alto a destra). Iniziamo.

A seguire (al centro a destra) ci concentriamo sul menù File. Ci saranno cinque elementi, New, Open, Save, un separatore e Exit. Useremo il metodo .add_command per aggiungere il comando. Tutto quello che dobbiamo realmente fare è chiamare il metodo con del testo (label =) e fornire una funzione di supporto per gestire il clic dell'utente sull'elemento. Per finire usiamo la funzione menubar.add_cascade per associare il menù alla barra.

Notate che il comando Exit usa root.quit per terminare il programma. Nessuna funzione di supporto è necessaria. A seguire facciamo lo stesso per i menù Edit e Help.

Osservate l'istruzione "tearoff=0" di ciascun gruppo di menù. Se cambiaste "0" con "1" il menù sarebbe circondato da una riga tratteggiata e in caso di trascinamento si "separerebbe" creando una finestra propria. Benché questo possa risultare utile in alcune circostanze, in questo caso è da evitare.

Alla fine, ma non per importanza, dobbiamo posizionare il menù. Non operiamo un semplice posizionamento tramite la funzione .grid(). Ricorriamo invece alla funzione parent.config (in basso a destra).

Tutto questo si trova nella routine BuildWidgets. Ora (prossima pagina, in alto a destra) dobbiamo aggiungere una cornice generica e impostare l'istruzione return prima di passare alla funzione PlaceWidgets.

Per finire (prossima pagina, in basso a destra) dobbiamo creare tutte le funzioni di supporto definite precedentemente. Per questa demo, tutto quello che faremo è stampare qualcosa nel terminale usato per lanciare il programma.

Questo è quanto. Salvate ed

```
def BuildWidgets(self,master):
    frame = Frame(master)
    #=============================
    #       MENU STUFF
    #=============================
    # Create the menu bar
    self.menubar = Menu(master)
```

```
# Create the File Pull Down, and add it to the menu bar
  filemenu = Menu(self.menubar, tearoff = 0)
  filemenu.add_command(label = "New", command = self.FileNew)
  filemenu.add_command(label = "Open", command = self.FileOpen)
  filemenu.add_command(label = "Save", command = self.FileSave)
  filemenu.add_separator()
  filemenu.add_command(label = "Exit", command = root.quit)
  self.menubar.add_cascade(label = "File", menu = filemenu)
```

```
# Create the Edit Pull Down
editmenu = Menu(self.menubar, tearoff = 0)
editmenu.add_command(label = "Cut", command = self.EditCut)
editmenu.add_command(label = "Copy", command = self.EditCopy)
editmenu.add_command(label = "Paste", command = self.EditPaste)
self.menubar.add_cascade(label = "Edit", menu = editmenu)
# Create the Help Pull Down
helpmenu = Menu(self.menubar, tearoff=0)
helpmenu.add_command(label = "About", command = self.HelpAbout)
self.menubar.add_cascade(label = "Help", menu = helpmenu)
```

```
  # Now, display the menu
master.config(menu = self.menubar)
  #=========================================
  #           End of Menu Stuff
  #=========================================
```

eseguite il programma. Fate clic su ciascuna voce dei menù (lasciando File->Exit per ultimo).

Ora (in basso) ci occupiamo del combo box. Salvate il file come widgetdemo2b.py e iniziamo. Gli import, la definizione della classe e le funzioni def __init__ sono gli stessi, come nella parte in basso del programma. Aggiungeremo due righe alla funzione DefineVars. Potete commentare l'istruzione "pass" o cancellarla e inserire il codice seguente (ho incluso la riga di definizione per chiarezza).

Prima definiamo un'etichetta che abbiamo creato prima. Quindi definiamo la casella combinata. Usiamo "ttk.Combobox", definiamo il genitore e impostiamo l'altezza a 19, la larghezza a 20 e textvariable a "self.cmbo1Val". Ricordate che abbiamo configurato textvariable nell'ultimo articolo, ma se l'aveste dimenticato... il suo testo cambia sincronizzandosi con il valore del combo box. Lo definiamo in DefineVars come un oggetto di tipo StringVar. Quindi carichiamo i valori tra cui vogliamo che l'utente scelga, definiti anche questi in DefineVars. Finiamo accoppiando l'evento virtuale ComboboxSelected alla funzione cmbotest che vedremo a breve.

```
        self.f1 = Frame(frame,
                        relief = SUNKEN,
                        borderwidth = 2,
                        width = 500,
                        height = 100
                        )

        return frame
```

Quindi ci occupiamo (come abbiamo fatto più volte) del posizionamento degli altri widget.

```
    def PlaceWidgets(self,master):
        frame = master
        frame.grid(column = 0, row = 0)

        self.f1.grid(column = 0,
                     row = 0,
                     sticky = 'nsew'
                     )
```

```
    def DefineVars(self):
        self.cmbo1Val = StringVar()
        self.c1Vals = ['None','Option 1','Option 2','Option 3']
```

Dopo la definizione di self.f1 in BuildWidgets e prima della riga "return frame" inserite il codice seguente.

```
        # Combo Box
        self.lblcb = Label(self.f1, text = "Combo Box: ")
        self.cmbo1 = ttk.Combobox(self.f1,
                                  height = "19",
                                  width = 20,
                                  textvariable = self.cmbo1Val
                                 )
        self.cmbo1['values'] = self.c1Vals
        # Bind the virtual event to the callback
        self.cmbo1.bind("<<ComboboxSelected>>",self.cmbotest)
```

```
    def FileNew(self):
        print "Menu - File New"

    def FileOpen(self):
        print "Menu - File Open"

    def FileSave(self):
        print "Menu - File Save"

    def EditCut(self):
        print "Menu - Edit Cut"

    def EditCopy(self):
        print "Menu - Edit Copy"

    def EditPaste(self):
        print "Menu - Edit Paste"

    def HelpAbout(self):
        print "Menu - Help About"
```

Proseguiamo posizionando il combo box e l'etichetta nel modulo (in alto a destra).

Salvate tutto e testate.

Ora salvate come widgetdemo2c.py e passiamo alla barra separatrice. Questa è davvero MOLTO semplice. Nonostante l'ultimo tkinter ne fornisca un widget, non sono mai riuscito a farlo funzionare. Ecco una semplice soluzione. Usiamo una cornice con altezza 2. Gli unici cambiamenti al programma saranno la definizione della cornice in BuildWidgets, dopo l'istruzione di associazione del combo box, e il posizionamento della cornice stessa, con la funzione PlaceWidgets. Quindi, in BuildWidgets inseriamo le righe seguenti (mostrate al centro a destra)...

Ancora una volta, tutto questo lo avete già visto. Salvate e testate. Probabilmente dovrete espandere la finestra più in alto per vedere il separatore, ma divverrà più chiaro nel prossimo passo. Salvate come widgetdemo2d.py perché ora tocca allo spin box.

Sotto DefineVars aggiungete la riga seguente...

```
self.spinval = StringVar()
```

Fino ad ora sapete che questa serve a recuperare il valore in qualunque momento vogliamo. Proseguiamo aggiungendo un po' di codice alla funzione BuildWidgets, proprio prima della riga "return frame" (in basso a destra).

Qui definiamo l'etichetta e il widget spin box. Quella dello spin box è la seguente:

```
ourwidget = Spinbox(parent,low value, high value, width, textvariable, wrap)
```

Il valore inferiore (low value) dovrebbe essere chiamato "from_" poiché la parola "from" è un termine chiave e usarlo in questo contesto porterebbe a confusione. I valori "from_" e "to" devono essere definiti come float. In questo caso vogliamo 1 come valore inferiore e 10 come valore superiore. Per finire, l'opzione wrap dice che se il valore è (nel nostro caso) 10 e l'utente fa clic sulla freccia rivolta in alto, si passa al valore inferiore, e così via. Stessa cosa per il valore inferiore. Se l'utente fa clic sulla freccia rivolta verso il basso e il valore è 1, si passa al valore 10. Se impostate "wrap=False", il widget si interrompe agli estremi.

```
        self.lblcb.grid(column = 0,row = 2)
        self.cmbo1.grid(column = 1,
                        row = 2,
                        columnspan = 4,
                        pady = 2
                        )
```

E per finire inseriamo la funzione di supporto che semplicemente stampa nel terminale ciò che l'utente seleziona.

```
    def cmbotest(self,p1):
        print self.cmbo1Val.get()
```

```
        self.fsep = Frame(self.f1,
                          width = 140,
                          height = 2,
                          relief = RIDGE,
                          borderwidth = 2
                          )
```

E in PlaceWidgets inserite questo...

```
        self.fsep.grid(column = 0,
                       row = 3,
                       columnspan = 8,
                       sticky = 'we',
                       padx = 3,
                       pady = 3
                       )
```

```
    self.lblsc = Label(self.f1, text = "Spin Control:")
    self.spin1 = Spinbox(self.f1,
                         from_ = 1.0,
                         to = 10.0,
                         width = 3,
                         textvariable = self.spinval,
                         wrap=True
                         )
```

Ora posizioneremo i widget in PlaceWidgets (in basso).

Ancora, questo è tutto. Salvate ed eseguite. Ora il separatore sarà evidente.

Salvate come widgetdemo2e.py e passiamo alle barre di progresso.

Ancora, abbiamo bisogno di definire alcune variabili, quindi nella funzione DefineVars aggiungiamo il codice seguente...

```
self.spinval2 = StringVar()
self.btnStatus = False
self.pbar2val = StringVar()
```

Dovrebbe essere abbastanza ovvio la funzione delle due StringVar. Discuteremo di "self.btnStatus" a breve. Per il momento definiamo i widget per questa sezione di BuildWidgets (a destra).

Anche questa va prima della riga "return frame". Ciò che stiamo facendo è impostare una cornice nella quale inserire i widget. Quindi configuriamo due etichette come guide. Poi definiamo la prima barra di progresso. Qui le uniche cose che potrebbero sembrare strane sono length, mode e maximum. Length è la dimensione in pixel della barra. Maximum è il valore più grande visualizzabile. In questo caso è 100 visto che mostreremo una percentuale. Mode in questo caso è "indeterminate". Ricordate, usiamo questa modalità quando non conosciamo il reale progresso dell'azione e vogliamo semplicemente notificare che qualcosa sta avvenendo.

Quindi aggiungiamo un pulsante (lo avete già fatto prima), un'altra etichetta, un'altra barra di progresso e un altro spin box. Il valore mode per la seconda barra di progresso è "determinate". Useremo il widget spin box per impostare la "percentuale" di completamento. Quindi aggiungete le righe seguenti (pagina seguente, in alto a sinistra) nella funzione PlaceWidgets.

Per finire aggiungiamo due

```
        self.lblsc.grid(column = 0, row = 4)
        self.spin1.grid(column = 1,
                        row = 4,
                        pady = 2
                        )
```

```
        #=======================================
        # Progress Bar Stuff
        #=======================================
        self.frmPBar = Frame(self.f1,
                             relief = SUNKEN,
                             borderwidth = 2
                             )

        self.lbl0 = Label(self.frmPBar,
                          text = "Progress Bars"
                          )
        self.lbl1 = Label(self.frmPBar,
                          text = "Indeterminate",
                          anchor = 'e'
                          )
        self.pbar = ttk.Progressbar(self.frmPBar,
                          orient = HORIZONTAL,
                          length = 100,
                          mode = 'indeterminate',
                          maximum = 100
                          )
        self.btnptest = Button(self.frmPBar,
                          text = "Start",
                          command = self.TestPBar
                          )
        self.lbl2 = Label(self.frmPBar,
                          text = "Determinate"
                          )
        self.pbar2 = ttk.Progressbar(self.frmPBar,
                          orient = HORIZONTAL,
                          length = 100,
                          mode = 'determinate',
                          variable = self.pbar2val
                          )
        self.spin2 = Spinbox(self.frmPBar,
                          from_ = 1.0,
                          to = 100.0,
                          textvariable = self.spinval2,
                          wrap = True,
                          width = 5,
                          command = self.Spin2Do
                          )
```

funzioni per controllare le barre di progresso (in basso a destra).

La funzione TestPBar controlla quella indeterminata. In pratica, avviamo e fermiamo un timer interno preconfigurato nella barra di progresso. La riga "self.pbar.start(10)" imposta il timer a 10 millisecondi. Questo valore farà muovere la barra abbastanza velocemente. Sentitevi liberi di provarne altri. La funzione Spin2Do semplicemente imposta la barra di progresso al valore scelto nel widget spin box. Lo stamperemo anche nel terminale.

Queste sono tutte le modifiche. Salvate e provate.

Ora salvate come widgetdemo2f.py e occupiamoci del widget notebook a schede. In BuildWidgets inserite il codice seguente (in basso) prima della riga "return frame"...

Osserviamo cosa abbiamo fatto. Prima abbiamo definito una cornice per il widget notebook. Poi definiamo il widget. Tutte le opzioni sono quelle già viste. Quindi definiamo due cornici

```
        #=======================================
        #               NOTEBOOK
        #=======================================
        self.nframe = Frame(self.f1,
                            relief = SUNKEN,
                            borderwidth = 2,
                            width = 500,
                            height = 300
                            )
        self.notebook = ttk.Notebook(self.nframe,
                                     width = 490,
                                     height = 290
                                     )
        self.p1 = Frame(self.notebook)
        self.p2 = Frame(self.notebook)
        self.notebook.add(self.p1,text = 'Page One')
        self.notebook.add(self.p2,text = 'Page Two')
        self.lsp1 = Label(self.p1,
                          text = "This is a label on
page number 1",
                          padx = 3,
                          pady = 3
                          )
```

```
        # Progress Bar
        self.frmPBar.grid(column = 0,
                          row = 5,
                          columnspan = 8,
                          sticky = 'nsew',
                          padx = 3,
                          pady = 3
                          )
        self.lbl0.grid(column = 0, row = 0)
        self.lbl1.grid(column = 0,
                       row = 1,
                       pady = 3
                       )
        self.pbar.grid(column = 1, row = 1)
        self.btnptest.grid(column = 3, row = 1)
        self.lbl2.grid(column = 0,
                       row = 2,
                       pady = 3
                       )
        self.pbar2.grid(column = 1, row = 2)
        self.spin2.grid(column = 3, row = 2)
```

```
    def TestPBar(self):
        if self.btnStatus == False:
            self.btnptest.config(text="Stop")
            self.btnStatus = True
            self.pbar.start(10)
        else:
            self.btnptest.config(text="Start")
            self.btnStatus = False
            self.pbar.stop()

    def Spin2Do(self):
        v = self.spinval2.get()
        print v
        self.pbar2val.set(v)
```

chiamate self.p1 e self.p2 che fungono da pagine. Le due righe successive (self.notebook.add) associano le cornici al widget notebook ottenendo una scheda. Impostiamo anche il testo per le schede. Per finire mettiamo un'etichetta sulla pagina numero uno. Ne metteremo una sulla pagina numero due quando posizioneremo i controlli, giusto per diletto.

Nella funzione PlaceWidgets inserite il codice seguente (in basso).

L'unica cosa che potrebbe apparire strana è l'etichetta di pagina due. Abbiamo combinato la definizione e il posizionamento nella griglia nello stesso comando. Lo abbiamo fatto nell'applicazione demo dell'ultima volta.

Questo è quanto. Salvate ed eseguite.

Come sempre, tutto il codice dell'applicazione completa si trova su pastebin, all'indirizzo http://pastebin.com/qSPkSNU1.

Divertitevi. La prossima volta ci occuperemo ancora di database.

```
        self.nframe.grid(column = 0,
                         row = 6,
                         columnspan = 8,
                         rowspan = 7,
                         sticky = 'nsew'
                         )
        self.notebook.grid(column = 0,
                           row = 0,
                           columnspan = 11,
                           sticky = 'nsew'
                           )
        self.lsp1.grid(column = 0,row = 0)
        self.lsp2 = Label(self.p2,
                          text = 'This is a label on PAGE 2',
                          padx = 3,
                          pady = 3
                          ).grid(
                                 column = 0,
                                 row = 1
                                 )
```

Programmare in Python - Parte 29

Un po' di tempo fa, mi fu chiesto di convertire in SQLite un database MySQL. Cercando su Internet una soluzione veloce e facile (e gratuita), non trovai nulla di utile per la mia versione di MySQL. Così decisi di fare "da me".

Il programma MySQL Administrator vi permette di creare una copia del database come semplice file di testo. Molti visualizzatori SQLite permettono di ricreare il database partendo da un file di questo tipo. Però, ci sono molte cose che MySQL supporta mentre SQLite no. Così questo mese scriveremo un programma di conversione che legge un file di istruzioni sql e crea la versione SQLite.

Iniziamo dando un'occhiata al file. Consiste di una sezione che crea il database, di una per creare ciascuna tabella e di quella eventuale per i dati. (C'è un'opzione per esportare solo lo schema della tabella). In alto a destra c'è un esempio della sezione per creare la tabella.

La prima cosa di cui abbiamo bisogno è l'ultima riga. Tutto quello dopo le parentesi di chiusura deve essere eliminato. (SQLite non supporta il database InnoDB). In aggiunta a questo, SQLite non supporta la riga "PRIMARY KEY". In SQLite si imposta una chiave primaria usando "INTEGER PRIMARY KEY AUTOINCREMENT" alla definizione del campo. L'altra cosa che SQLite non digerisce è la parola chiave "unsigned".

Parlando di dati, l'istruzione "INSERT INTO" non è compatibile. Il problema quì è che SQLite non permette inserimenti multipli all'interno della stessa istruzione. Ecco un breve esempio preso da un file MySQL. Notate (a destra) che il marcatore di fine riga è un punto e virgola.

Durante la conversione ignoreremo anche le righe di commento e le istruzioni CREATE DATABASE e USE. Una volta ottenuto il file SQLite, useremo un programma simile a SQLite Database Browser per il processo di

```
DROP TABLE IF EXISTS `categoriesmain`;
CREATE TABLE `categoriesmain` (
  `idCategoriesMain` int(10) unsigned NOT NULL
auto_increment,
  `CatText` char(100) NOT NULL default '',
  PRIMARY KEY  (`idCategoriesMain`)
) ENGINE=InnoDB AUTO_INCREMENT=40 DEFAULT
CHARSET=latin1;
```

```
INSERT INTO `categoriesmain`
(`idCategoriesMain`,`CatText`) VALUES
 (1,'Appetizer'),
 (2,'Snack'),
 (3,'Barbecue'),
 (4,'Cake'),
 (5,'Candy'),
 (6,'Beverages');
```

Per renderlo compatibile, dobbiamo cambiare questo da singola istruzione complessa ad una serie di istruzioni più semplici come segue:

```
INSERT INTO `categoriesmain`
(`idCategoriesMain`,`CatText`) VALUES (1,'Appetizer');
INSERT INTO `categoriesmain`
(`idCategoriesMain`,`CatText`) VALUES (2,'Snack');
INSERT INTO `categoriesmain`
(`idCategoriesMain`,`CatText`) VALUES (3,'Barbecue');
INSERT INTO `categoriesmain`
(`idCategoriesMain`,`CatText`) VALUES (4,'Cake');
INSERT INTO `categoriesmain`
(`idCategoriesMain`,`CatText`) VALUES (5,'Candy');
INSERT INTO `categoriesmain`
(`idCategoriesMain`,`CatText`) VALUES (6,'Beverages');
```

creazione del database, delle tabelle e dei dati.

Iniziamo. Create una cartella per il progetto e un nuovo file python. Chiamatelo MySQL2SQLite.py.

In alto a destra c'è l'istruzione import, la definizione della classe e la routine __init__.

Poiché sarà un programma da riga di comando dobbiamo creare l'istruzione "if __name__", un gestore degli argomenti e una routine d'aiuto (che informi l'utente su come utilizzare il programma). Questa andrà inserita alla fine. Tutto il resto del codice creato andrà prima:

```
def error(message):
      print >> sys.stderr,
str(message)
```

In basso c'è il gestore che si occupa di stampare le istruzioni d'uso.

La funzione DoIt() viene chiamata se il programma è eseguito da solo dalla riga di comando, come previsto. Comunque, se volessimo in seguito aggiungerlo come libreria ad un altro programma, potremmo semplicemente usare la classe. Quì impostiamo un numero di variabili per essere sicuri che tutto funzioni correttamente. Il codice mostrato in basso a destra valuta gli argomenti passati e li prepara per le routine principali.

Quando avviamo il programma dobbiamo fornire almeno due variabili. Sono il file di Input e quello di Output. All'utente saranno disponibili altre tre opzioni: una è la modalità debug, affinché ci si renda

```
#!/usr/bin/env python
#=====================================
# MySQL2SQLite.py
#=====================================
#             IMPORTS
import sys
#=====================================

#=====================================
#     BEGIN CLASS MySQL2SQLite
#=====================================
class MySQL2SQLite:
    def __init__(self):
        self.InputFile = ""
        self.OutputFile = ""
        self.WriteFile = 0
        self.DebugMode = 0
        self.SchemaOnly = 0
        self.DirectMode = False
```

```
def DoIt():
    #=======================================
    #            Setup Variables
    #=======================================
    SourceFile = ''
    OutputFile = ''
    Debug = False
    Help = False
    SchemaOnly = False
    #=======================================
```

```
if len(sys.argv) == 1:
    usage()
else:
    for a in sys.argv:
        print a
        if a.startswith("Infile="):
            pos = a.find("=")
            SourceFile = a[pos+1:]
        elif a.startswith("Outfile="):
            pos = a.find("=")
            OutputFile = a[pos+1:]
        elif a == 'Debug':
            Debug = True
        elif a == 'SchemaOnly':
            SchemaOnly = True
        elif a == '-Help' or a == '-H' or a == '-?':
            Help = True
    if Help == True:
        usage()
    r = MySQL2SQLite()
    r.SetUp(SourceFile,OutputFile,Debug,SchemaOnly)
    r.DoWork()
```

conto di cosa sta avvenendo man mano che il programma procede; un'opzione per creare solo le tabelle e non i dati e una per chiedere aiuto. La riga di comando "normale" per avviare il programma assomiglia a questa:

```
MySQL2SQLite Infile=Foo
Outfile=Bar
```

Dove "Foo" è il nome del file MySQL e "Bar" è il nome del file SQLite che vogliamo il programma crei.

Potete usare anche questa versione:

```
MySQL2SQLite Infile=Foo
Outfile=Bar Debug SchemaOnly
```

che aggiunge l'opzione per mostrare i messaggi di debug ed esporta solo le tabelle e non i dati.

Per finire se l'utente chiede aiuto, si passa semplicemente alla sezione con le istruzioni d'uso.

Prima di continuare diamo un'occhiata a come vengono gestiti gli argomenti nella riga di comando.

Quando un utente inserisce il nome del programma nel terminale, il sistema operativo tiene traccia dell'informazione immessa e la passa al programma che valuta l'eventuale presenza di opzioni. Se non è stata inserita nessuna opzione (o argomento) il suo numero è uno, cioè il nome dell'applicazione, nel nostro caso MySQL2SQLite.py. Possiamo accedere a questi argomenti chiamando il comando sys.arg. Se il conteggio è maggiore di uno, usiamo un ciclo for per leggerle. Controlleremo gli argomenti uno alla volta. Alcuni programmi richiedono di inserirli in uno specifico ordine. Usando l'approccio del ciclo for, questi possono essere inseriti in qualunque ordine. Se l'utente non fornisce alcun argomento o usa l'argomento aiuto mostreremo la schermata sull'uso. In alto c'è la relativa routine.

```
def usage():
    message = (
       '================================================================\n'
        'MySQL2SQLite - A database converter\n'
        'Author: Greg Walters\n'
        'USAGE:\n'
        'MySQL2SQLite Infile=filename [Outfile=filename] [SchemaOnly] [Debug] [-H-Help-?\n'

        '   where\n'
        '         Infile is the MySQL dump file\n'
        '         Outfile (optional) is the output filename\n'
        '            (if Outfile is omitted, assumed direct to SQLite\n'
        '         SchemaOnly (optional) Create Tables, DO NOT IMPORT DATA\n'
        '         Debug (optional) - Turn on debugging messages\n'
        '         -H or -Help or -? - Show this message\n'
        'Copyright (C) 2011 by G.D. Walters\n'
        '================================================================\n'
       )
    error(message)
    sys.exit(1)

if __name__ == "__main__":
    DoIt()
```

Continuando, una volta terminato il controllo degli argomenti istanziamo la classe, chiamiamo la funzione SetUp, che riempie alcune variabili e quindi chiamiamo la routine DoWork. Inizieremo dalla classe (mostrata nella prossima pagina, in basso a destra).

Questa (pagina seguente, in basso a destra) contiene la definizione e la funzione __init__. Quì definiremo le variabili di cui avremo bisogno più avanti. Ricordate che prima di chiamare la funzione DoWork bisogna chiamare SetUp. Quì assegneremo i valori corretti alle variabili vuote. Abbiamo anche la possibilità di non scrivere in un file, utile a scopi di debug, e di scrivere semplicemente lo schema senza i dati, utile per replicare in un nuovo database solo la struttura.

Iniziamo aprendo il file SQL e impostiamo alcune variabili interne. Definiamo anche alcune stringhe contenenti testo usato da usare più volte. Ovviamente se dovremo scrivere nel file di output lo dovremo prima aprire e iniziare l'intero processo. Leggeremo ciascuna riga del file d'ingresso, la processeremo e scriveremo il testo potenziale nel file di output. Usiamo un ciclo while forzato per la lettura di ciascuna riga, con un comando di interruzione quando non è rimasto nulla nel file in ingresso. Usiamo f.readline() per recuperare la riga di lavoro e assegniamo il suo contenuto alla variabile "line". Alcune righe possono essere tranquillamente ignorate. A questo servono le istruzioni if/elif seguite dall'istruzione pass (in basso).

Possiamo finalmente smettere di occuparci di cosa ignorare e passare a quelle utili. Il processo inizierà nel momento in cui la riga conterrà Create Table. Ricordate che abbiamo assegnato a CT il valore "Create Table". Quì (in alto a destra), impostiamo la

```
    def SetUp(self, In, Out = '', Debug = False, Schema = 0):
        self.InputFile = In
        if Out == '':
            self.writeFile = 0
        else:
            self.WriteFile = 1
            self.OutputFile = Out
        if Debug == True:
            self.DebugMode = 1
        if Schema == 1:
            self.SchemaOnly = 1
```

Ora, ci occuperemo della funzione DoWork, dove avviene la "magia".

```
    def DoWork(self):
        f = open(self.InputFile)
        print "Starting Process"
        cntr = 0
        insertmode = 0
        CreateTableMode = 0
        InsertStart = "INSERT INTO "
        AI = "auto_increment"
        PK = "PRIMARY KEY "
        IPK = " INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL"
        CT = "CREATE TABLE "
        # Begin
        if self.WriteFile == 1:
            OutFile = open(self.OutputFile,'w')
```

```
    while 1:
        line = f.readline()
        cntr += 1
        if not line:
            break
        # Ignore blank lines, lines that start with
"--" or comments (/*!)
        if line.startswith("--"): #Comments
            pass
        elif len(line) == 1: # Blank Lines
            pass
        elif line.startswith("/*!"): # Comments
            pass
        elif line.startswith("USE"):
            #Ignore USE lines
            pass
        elif line.startswith("CREATE DATABASE "):
            pass
```

```
#====================================
#     BEGIN CLASS MySQL2SQLite
#====================================
class MySQL2SQLite:
    def __init__(self):
        self.InputFile = ""
        self.OutputFile = ""
        self.WriteFile = 0
        self.DebugMode = 0
        self.SchemaOnly = 0
```

variabile "CreateTableMode" uguale a uno, così da sapere cosa stiamo facendo, poiché la definizione di ciascun campo è su una riga separata. Quindi prendiamo la riga, rimuoviamo il carattere di ritorno e la teniamo pronta per scriverla nel file in uscita, e, se richiesto, la scriviamo.

Ora (al centro destra) dobbiamo iniziare ad occuparci di ciascuna riga all'interno del blocco che crea la tabella, gestendo ciascuna riga per fare contento SQLite. Ci sono molte cose che SQLite non supporta. Diamo ancora un'occhiata all'istruzione Create Table di MySQL.

Una cosa che assolutamente da problemi a SQLite è l'intera ultima riga dopo le parentesi di chiusura. Un'altra è la riga subito sopra, quella con Primary Key. Un'altra cosa è la parola chiave unsigned nella seconda riga. Sarà richiesto un po' di codice (in basso) per superare questi problemi, ma possiamo farcela.

Prima (terzo inferiore a destra) controlliamo se la riga contiene "auto increment". Assumeremo che questa sia la riga della chiave primaria, come accade nel 98,6% dei casi, ma potrebbe non esserlo. Comunque noi resteremo sul semplice. A seguire controlliamo se la riga inizia con ") ". Questo significa che si tratta dell'ultima riga del blocco. In questo caso inseriamo semplicemente una stringa nella variabile "newline" per chiudere propriamente l'istruzione, azzeriamo la variabile CreateTableMode e, se stiamo scrivendo in un file, scriviamo.

Ora (in basso a destra) usiamo l'informazione trovata a proposito della parola chiave auto incremento. Prima eliminiamo dalla riga gli spazi inutili, quindi cerchiamo la posizione della stringa " int(" (dando per certo che ci sia) all'interno della riga. La sostituiremo con la frase " INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL". A SQLite non interessa la lunghezza dell'intero. Ancora, scriviamo sul file se se richiesto.

Ora cercheremo la frase "PRIMARY KEY ". Notate lo spazio extra alla fine; è voluto. In caso affermativo,

```
    elif line.startswith(CT):
        CreateTableMode = 1
        l1 = len(line)
        line = line[:l1-1]
        if self.DebugMode == 1:
            print "Starting Create Table"
            print line
        if self.WriteFile == 1:
            OutFile.write(line)
```

```
CREATE TABLE `categoriesmain` (
  `idCategoriesMain` int(10) unsigned NOT NULL auto_increment,
  `CatText` char(100) NOT NULL default '',
  PRIMARY KEY  (`idCategoriesMain`)
) ENGINE=InnoDB AUTO_INCREMENT=40 DEFAULT CHARSET=latin1;
```

```
    p1 = line.find(AI)
    if line.startswith(") "):
        CreateTableMode = 0
        if self.DebugMode == 1:
            print "Finished Table Create"
        newline = ");\n"
        if self.WriteFile == 1:
            OutFile.write(newline)
            if self.DebugMode == 1:
                print "Writing Line {0}".format(newline)
```

```
    elif p1 != -1:
        # Line is primary key line
        l = line.strip()
        fnpos = l.find(" int(")
        if fnpos != -1:
            fn = l[:fnpos]
        newline = fn + IPK #+ ",\n"
        if self.WriteFile == 1:
            OutFile.write(newline)
            if self.DebugMode == 1:
                print "Writing Line {0}".format(newline)
```

```
    elif CreateTableMode == 1:
        # Parse the line...
        if self.DebugMode == 1:
            print "Line to process — {0}".format(line)
```

ignoreremo la riga.

```
elif
line.strip().startswith(PK):
      pass
```

Proseguite (in alto a destra) cercando la stringa " unsigned " (anche qui mantenete gli spazi extra) e sostituitela da con " ".

Così termina la routine per creare la tabella. Ora (in basso) ci spostiamo alle istruzioni per l'inserimento dei dati. La variabile InsertStart è la frase "INSERT INTO ". La cerchiamo perché MySQL permette molteplici istruzioni di inserimento in un singolo comando, ma SQLite no. Dobbiamo creare istruzioni separate per ciascun blocco di dati. Impostiamo la variabile "insertmode" a 1, inseriamo "INSERT INTO {Table} {Fieldlist} VALUES (" in una variabile riutilizzabile (che chiameremo preambolo), e continuiamo.

Controlliamo per vedere se dobbiamo soltanto lavorare sullo schema. In caso affermativo, possiamo ignorare in sicurezza le sezioni delle istruzioni di inserimento. In caso contrario, dobbiamo occuparcene.

```
elif self.SchemaOnly == 0:
      if insertmode == 1:
```

Controlliamo per vedere se c'è ");" o ")," nella riga. In caso di ");", questa sarà l'ultima riga del gruppo di inserimento.

```
posx = line.find(");")
pos1 = line.find("),")
l1 = line[:pos1]
```

Questa riga controlla la presenza di apici singoli preceduti dal carattere di escape e li sostituisce.

```
line =
line.replace("\\'","''")
```

Se abbiamo un'istruzione di chiusura (");"), questa è la fine

```
    elif line.startswith(InsertStart):
        if insertmode == 0:
            insertmode = 1
            # Get tablename and field list here
            istatement = line
            # Strip CR/LF from istatement line
            l = len(istatement)
            istatement = istatement[:l-2]
```

```
    elif line.find(" unsigned ") != -1:
        line = line.replace(" unsigned ","  ")
        line = line.strip()
        l1 = len(line)
        line = line[:l1-1]
        if self.WriteFile == 1:
            OutFile.write("," + line)
            if self.DebugMode == 1:
                print "Writing Line {0}".format(line)
```

Altrimenti, ci occupiamo della riga.

```
    else:
        l1 = len(line)
        line = line.strip()
        line = line[:l1-4]
        if self.DebugMode == 1:
            print "," + line
        if self.WriteFile == 1:
            OutFile.write("," + line)
```

```
    if posx != -1:
        l1 = line[:posx+3]
        insertmode = 0
        if self.DebugMode == 1:
            print istatement + l1
            print "------------------------------"
        if self.WriteFile == 1:
            OutFile.write(istatement + l1+"\n")
```

Altrimenti,colleghiamoilpreamboloalvalore dell'istruzione e finiamo con un punto e virgola.

```
    elif pos1 != -1:
        l1 = line[:pos1+2]
        if self.DebugMode == 1:
            print istatement + l1 + ";"
        if self.WriteFile == 1:
            OutFile.write(istatement + l1 + ";\n")
```

dell'insieme di inserimento e possiamo creare l'istruzione per unire il preambolo al valore attuale dell'istruzione. Questo è mostrato nella pagina precedente, in basso a destra.

Tutto questo funziona (in alto a destra) se l'ultimo valore che abbiamo nell'istruzione di inserimento è una stringa tra doppi apici. Comunque se l'ultimo valore è di tipo numerico dobbiamo lavorare differentemente. Sarete in grado di capire ciò che stiamo facendo quì.

Per finire, chiudiamo il file di input e, se stiamo scrivendo in un file, chiudiamo anche quello di output.

```
f.close()
if self.WriteFile == 1:
    OutFile.close()
```

Una volta ottenuto il file convertito, potete usare SQLite Database Browser per ricreare la struttura del database e i dati.

Questo codice dovrebbe funzionare nel 90% dei casi. Potrebbe esserci qualcosa che abbiamo tralasciato a causa di altri motivi, quindi ecco il motivo della modalità di debug. Comunque, ho testato questo su più file e non ho avuto problemi.

Come al solito, il codice è su PasteBin all'indirizzo http://pastebin.com/cPvzNT7T.

Alla prossima.

Greg Walters è il proprietario della RainyDay Solutions, LLC, una società di consulenza in Colorado e programma dal 1972. Ama cucinare, fare escursioni, ascoltare musica e passare il tempo con la sua famiglia. Il suo sito web è www.thedesignatedgeek.com.

```
else:
    if self.DebugMode == 1:
        print "Testing line {0}".format(line)
    pos1 = line.find("),")
    posx = line.find(");")
    if self.DebugMode == 1:
        print "pos1 = {0}, posx = {1}".format(pos1,posx)
    if pos1 != -1:
        l1 = line[:pos1+1]
        if self.DebugMode == 1:
            print istatement + l1 + ";"
        if self.WriteFile == 1:
            OutFile.write(istatement + l1 + ";\n")
    else:
        insertmode = 0
        l1 = line[:posx+1]
        if self.DebugMode == 1:
            print istatement + l1 + ";"
        if self.WriteFile == 1:
            OutFile.write(istatement + l1 + ";\n")
```

Iniziare Python - Parte 30

Questo mese esploreremo un nuovo disegnatore di GUI, questa volta per Tkinter. Molte persone hanno problemi ad utilizzare Tkinter perché non offre uno strumento proprio. Benché vi abbia mostrato quanto sia semplice disegnare un'applicazione senza un designer, ora analizzeremo Page. In pratica si tratta di una versione di Visual TCL con supporto python predefinito. La versione corrente è la 3.2 che può essere trovata al seguente indirizzo http://sourceforge.net/projects/page/files/latest/download.

Prerequisiti

Avete bisogno di TCK/TK 8.5.4 o successivo, Python 2.6 o successivo, e pyttk che potete scaricare, se non lo avete già fatto, da http://pypi.python.org/pypi/pyttk. Probabilmente li possedete già tutti con la possibile eccezione di pyttk.

Installazione

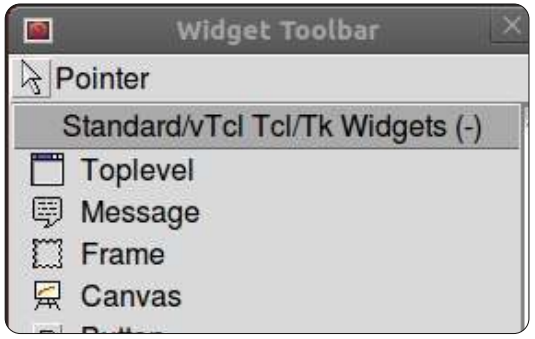
Non potete proprio chiedere una procedura di installazione più semplice. Estraete il contenuto dell'archivio in una cartella a vostra scelta. Entrate nella cartella ed eseguite lo script chiamato "configure". Questo creerà lo script lanciatore chiamato "page" che userete per fare tutto. Questo è quanto.

Imparare Page

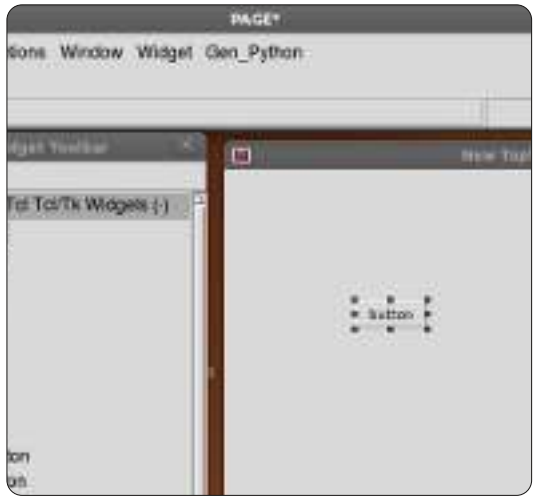
Quando avviate Page, vi troverete di fronte a tre finestre (form). Una è la "launch pad", una è quella con gli strumenti e una è l'editor degli attributi.

Per iniziare un nuovo progetto fare clic sul pulsante Toplevel nella finestra degli strumenti.

Verrà così creata la finestra principale. Potete spostarla ovunque nello schermo. Quindi, e a partire da



ora, fate clic su un widget nella finestra strumenti e ancora clic nella finestra principale dove lo si vuole posizionare.

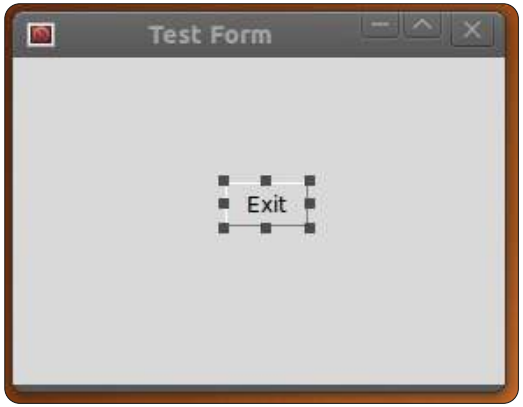


Per il momento inseriamo un pulsante. Fate clic sul pulsante Button nella finestra strumenti e quindi fate clic ovunque nella finestra principale.

Nella finestra del launch pad fate clic su Window e selezionate Attribute Editor (se non è già aperto). Il vostro unico pulsante dovrebbe già essere evidenziato, allora spostatelo nella finestra e quando rilascerete il pulsante del mouse vedrete che i valori di 'x position' e 'y position' nella finestra degli attributi cambieranno.

Qui possiamo configurare altri attributi come il testo sul pulsante (o sulla maggior parte dei widget), l'alias per il widget (il nome a cui faremo riferimento nel codice), il colore, il nome con cui lo chiameremo e via così. Verso il fondo alla finestra degli attributi c'è il campo text. In questo caso, si tratta del testo mostrato all'utente per il pulsante. Cambiamolo da "button" a "Exit". Notate come ora il pulsante riporti la scritta "Exit". Ora ridimensioniamo la finestra per mostrare solamente il pulsante che

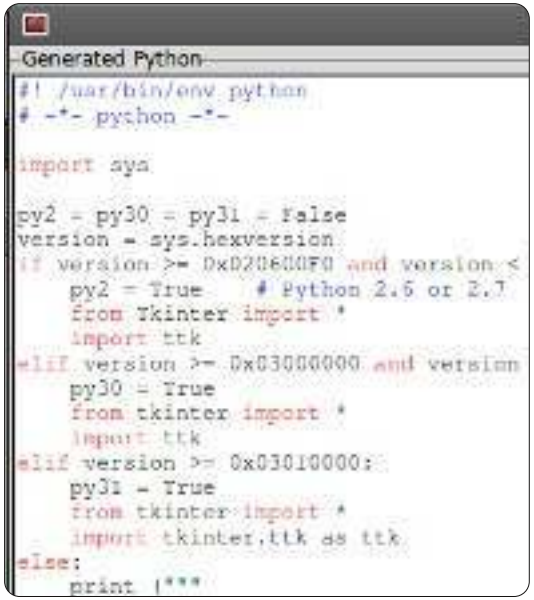
provvederemo a centrare.



Quindi fate clic nella finestra principale ovunque tranne che sul pulsante. La finestra degli attributi ora mostra gli attributi della finestra principale. Cercate il campo "title" e cambiatelo da "New Toplevel 1" a "Test Form".

Ora, prima di salvare il progetto, dobbiamo creare una cartella per contenere i nostri file. Create la cartella chiamata "PageProjects" dove volete nel disco. Adesso, nella finestra launch pad, selezionate File quindi Save As. Navigate fino alla cartella PageProjects e, nella finestra digitate TestForm.tcl e fate clic sul pulsante Salva. Notate che è stato salvato come file tcl, non python. Creeremo il file python successivamente.

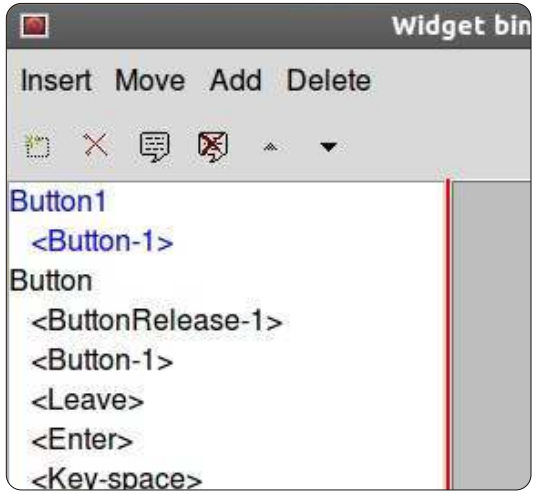
In launch pad trovate la voce di menu Gen_Python e fate clic. Selezionate Generate Python e comparirà una nuova finestra.



Page ha generato, come il nome suggerisce, il codice python e lo ha posizionato in una finestra visibile. In basso a questa finestra, ci sono tre pulsanti... Save, Run e Close.

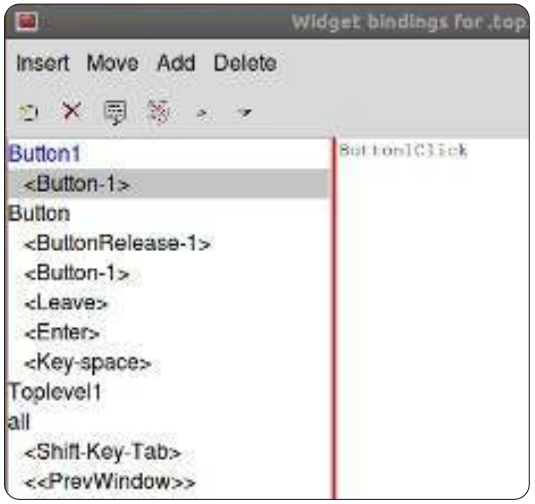
Fate clic su Save. Se, a questo punto, date un'occhiata alla cartella PageProjects, vedrete il file python (TestForm.py). Ora fate clic sul pulsante Run. In pochi secondi vedrete il progetto avviarsi. Il pulsante non è ancora connesso a nulla, così non succederà niente facendoci clic. Chiudete semplicemente la finestra con la X all'angolo. Ora chiudete la console python con il pulsante Close in basso a destra.

Nuovamente nella nostra finestra



principale evidenziate il pulsante Exit e fate clic destro. Selezionate "Bindings...". Sotto il menù ci sono una serie di pulsanti.

Il primo a sinistra vi permette di



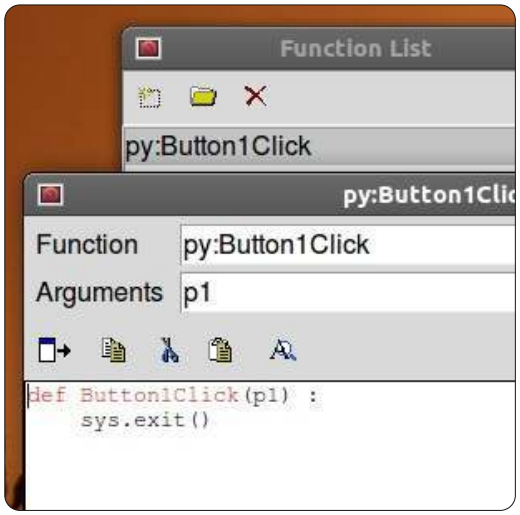
creare un nuovo legame. Fate clic su "Button-1". Questo ci permetterà di inserire il legame per il pulsante sinistro del mouse. Nella finestra a destra digitate "Button1Click".

Salvate e generate nuovamente il codice python. Scorrete verso la fine del file nella console python. Sopra il codice "class Test_Form" c'è la funzione che abbiamo proprio ora creato. Notate che per il momento viene semplicemente ignorata. Guardate ulteriormente in basso e vedrete il codice che crea e controlla il pulsante. Tutto è già pronto. Però dobbiamo ancora dire al pulsante cosa fare. Chiudete la console python e continuiamo.

Nel launch pad fate clic su Window quindi selezionate Function List. Qui scriveremo il metodo per chiudere la finestra.

Il primo pulsante a sinistra è

quello Add. Fate clic. Nel campo Function digitate "py:Button1Click" e nel campo Arguments inserite "p1" e cambiate il testo nel riquadro in basso in...




```
def Button1Click(p1):
    sys.exit()
```

Per finire, fate clic sul segno di spunta.

A seguire dobbiamo legare questa routine al pulsante. Selezionate il pulsante nella finestra, fate clic destro e selezionate "Bindings...". Come prima, fate clic sul pulsante più a sinistra della barra strumenti e selezionate Button-1. Questo è l'evento associato al clic del pulsante sinistro del mouse. Nel riquadro di testo a destra inserite "Button1Click". Assicuratevi di usare le stesse maiuscole/minuscole usate per la funzione appena creata. Fate clic sul segno di spunta sul lato destro.

Ora salvate e generate il codice python.

Dovreste vedere il seguente codice verso la fine, ma fuori alla classe Test_Form...

```
def Button1Click(p1) :
sys.exit()
```

E l'ultima linea della classe dovrebbe essere...

```
self.Button1.bind('<Button-1>',Button1Click)
```

Se eseguite il codice e fate clic sul pulsante Exit la finestra dovrebbe chiudersi correttamente.

Proseguire

Ora facciamo qualcosa di più complicato. Creeremo una demo mostrante i widget disponibili. Prima di tutto chiudiamo Page e riavviamolo. Proseguite creando un nuovo modulo Toplevel. Aggiungete due frame, uno sopra l'altro e espandeteli fino a occupare l'intera larghezza della finestra. Nel frame superiore posizionate un'etichetta e, usando l'editor degli attributi, cambiate il testo in "Buttons:". Quindi, aggiungete due pulsanti lungo il piano orizzontale. Cambiate il testo di quello sinistro in "Normal" e di quello destro in "Sunken". Mentre è selezionato il pulsante Sunken, cambiate l'attributo over relief in "sunken" e chiamatelo btnSunken. Per il pulsante "Normal" scegliete l'alias "btnNormal". Salvate

il progetto come "Demos.tcl".

Proseguite posizionando nel frame inferiore un'etichetta con scritto "Radio Buttons" e quattro pulsanti radio come nella immagine in basso. Per finire, posizionate un pulsante Exit sotto il frame inferiore.

Prima di lavorare sui legami, creiamo le funzioni per gestire i clic. Aprite Function List e create due funzioni. La prima dovrebbe essere chiamata btnNormalClicked e l'altra btnSunkenClicked. Assicuratevi di configurare i campi Arguments per includere p1. Ecco il codice che dovreste avere...

```
def btnNormalClicked(p1):
print "Normal Button Clicked"

def btnSunkenClicked(p1) :
print "Sunken Button Clicked"
```

Aggiungiamo i legami per i pulsanti. Per ciascun pulsante fate clic destro, selezionate ""Bindings..." e aggiungete, come prima, un collegamento alla funzione che abbiamo creato. Per il pulsante normale, dovrebbe essere "btnNormalClicked" e per quello incavato dovrebbe essere btnSunkenClicked. Salvate e generate il codice. Ora testando il programma con l'opzione Run della console python, facendo clic su qualunque pulsante, non vedrete accadere nulla. Comunque, chiusa l'applicazione, dovreste vedere l'output. Questo è normale per Page e se lo eseguite dalla riga di comando come normalmente si fa le cose dovrebbero funzionare come ci si aspetta.

Ora tocca ai pulsanti radio. Li abbiamo raggruppati in due insiemi. I primi due (Radio 1 e Radio 2) saranno l'insieme 1 e gli altri due saranno l'insieme 2. Fate clic su Radio 1 e nell'editor degli attributi impostate l'attributo value a zero e quello variable a "rbc1". Configurare variable per Radio 2 a "rbc1" e value a uno. Fate la stessa cosa per Radio 3 e Radio 4 ma per entrambi impostate variable a "rbc2". Se volete, potete gestire il clic dei pulsanti radio stampndo qualcosa nel terminale, ma per ora la cosa importante è che i due gruppi funzionino. Facendo clic su Radio 1 verrà deselezionato Radio 2 senza influenzare Radio 3 o Radio 4, stessa cosa per Radio 2 e così via.

Per finire dovreste creare una funzione per il pulsante Exit e legarlo al pulsante così come abbiamo fatto nel primo esempio.

Se avete seguito quello che abbiamo realizzato nelle altre applicazioni Tkinter, dovreste comprendere il codice mostrato in alto a destra. In caso contrario tornate indietro di qualche numero per una discussione completa.

Avrete notato come Page renda il processo di creazione molto più semplice che farlo da soli. Abbiamo solo scalfito la superficie di quello che Page può fare e inizieremo a fare qualcosa di più impegnativo la prossima volta.

```
def set_Tk_var():
# These are Tk variables passed to Tkinter and must
# be defined before the widgets using them are created.
global rbc1
rbc1 = StringVar()
global rbc2
rbc2 = StringVar()
def btnExitClicked(p1) :
sys.exit()
def btnNormalClicked(p1) :
print "Normal Button Clicked"
def btnSunkenClicked(p1) :
print "Sunken Button Clicked"
```

Il codice python lo potete trovare su pastebin all'indirizzo http://pastebin.com/qq0YVgTb.

Una nota prima di chiudere per questo mese. Potreste aver notato che ho saltato un paio di numeri. È dovuto al fatto che a mia moglie l'anno scorso è stato diagnosticato un cancro. Nonostante abbia cercato di non farmi crollare il mondo addosso, alcune cose sono accadute. Una di queste è che il mio vecchio dominio/sito web all'indirizzo ~~www.thedesignatedgeek.com~~ è stato prima chiuso per mancato rinnovo e poi venduto a mia insaputa. Ho predisposto www.thedesignatedgeek.net con tutto il vecchio materiale. Devo sistemare tutte le vecchie cose. Dovrò lavorare duramente il mese prossimo per rimettermi in carreggiata.

Ci vediamo la prossima volta.

Greg Walters è il proprietario della RainyDay Solutions, LLC, una società di consulenza in Aurora, Colorado e programma dal 1972. Ama cucinare, fare escursioni, ascoltare musica e passare il tempo con la sua famiglia. Il suo sito web è www.thedesignatedgeek.net.

ubuntu uk podcast

Il podcast di Ubuntu tratta di tutte le ultime novità e di argomenti riguardanti gli utenti Ubuntu Linux e i fan del software libero in generale. Lo show piacerà ai nuovi utenti così come a quelli più navigati. Le nostre discussioni riguardano lo sviluppo di Ubuntu senza troppi tecnicismi. Siamo abbastanza fortunati da avere in studio alcuni ospiti importanti che ci racconteranno in anteprima degli ultimi eccitanti sviluppi a cui stanno lavorando, in maniera comprensibile! Parliamo inoltre della comunità Ubuntu e di cosa si occupa.

Lo spettacolo è presentato da membri della comunità Ubuntu Linux inglese. Basandosi sul Codice di Condotta Ubuntu è adatta a tutti.

Lo show è trasmesso in diretta ogni due settimane nella serata di martedì (ora britannica) ed è disponibile per il download il giorno seguente.

podcast.ubuntu-uk.org

Iniziare Python - Parte 31

Dopo il nostro ultimo incontro dovreste avere una discreta idea di come usare Page. Se non è così, per favore leggete l'articolo dello scorso mese. Continueremo questa volta creando una applicazione per elencare i file tramite GUI. L'obbiettivo è creare un'applicazione grafica che ricorsivamente sfoglia una directory, ricercando file con un insieme di estensioni definite e mostrando il risultato in una vista ad albero. Per questo esempio cercheremo file di tipo media con estensione ".avi", ".mkv", ".mv4", "mp3" e ".ogg".

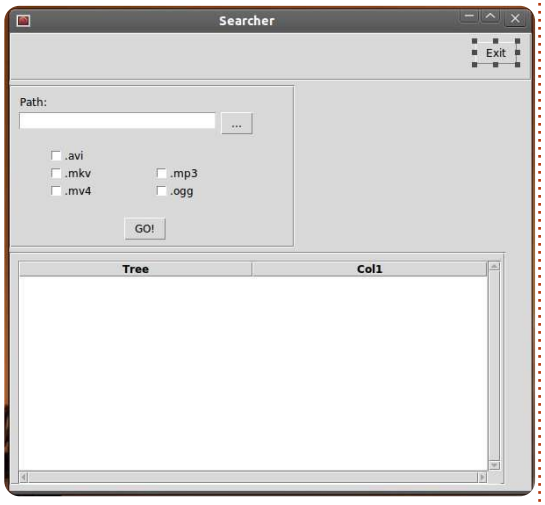
Stavolta il testo può sembrare un po' succinto nella porzione del progetto. Tutto quello che sto per fare è dare delle indicazioni per la disposizione dei widget e dei valori e attributi necessari, come queste...

```
Widget

Attribute: Value
```

Metterò tra apici le stringhe di testo solo quando è necessario. Per esempio per uno dei bottoni, il testo deve essere impostato a "...".

Ecco come dovrebbe essere la GUI della nostra applicazione...



Come si può vedere, c'è il modulo principale, un pulsante per uscire, una casella per l'inserimento di testo con un pulsante che richiamerà una finestra di dialogo per la directory, 5 caselle di scelta per la selezione delle estensioni, un pulsante "GO!" per iniziare effettivamente l'elaborazione e una vista ad albero per mostrare il risultato.

Quindi, cominciamo. Avviamo Page e creiamo un nuovo widget di primo livello. Utilizzando l'editor degli Attributi assegniamo i seguenti attributi.

```
Alias: Searcher
Title: Searcher
```

Assicurarsi di salvare spesso. Quando si salva il file, farlo con il nome "Searcher". Ricordate, Page mette l'estensione .tcl per noi e quando infine si genere il codice python, lo salverà nello stessa cartella.

Successivamente aggiungiamo un frame. Dovrebbe andare in cima a quello principale. Assegniamo gli attributi come indicato di seguito.

```
Width: 595
Height: 55
x position: 0
y position: 0
```

In questo frame, aggiungiamo un pulsante. Questo sarà il nostro pulsante per l'Uscita.

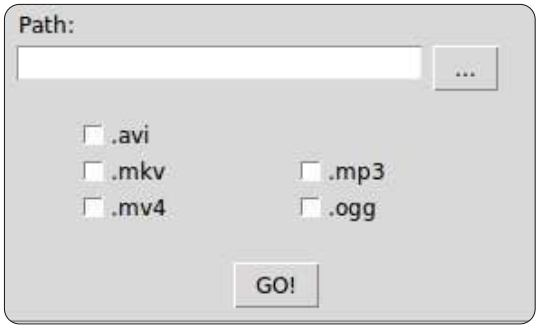
```
Alias: btnExit
Text: Exit
```

Spostiamolo verso il centro o vicino al lato destro del frame. Io imposto il mio con X 530 e Y 10.

Creiamo un altro frame.

```
Width: 325
Height: 185
y position: 60
```

Ecco qua come apparirà il frame, per dare una guida e andare avanti in questa sezione.



Aggiungiamo una etichetta in questo frame e impostiamo l'attributo del testo a "Path:". Spostiamolo vicino all'angolo superiore sinistro del frame.

Nello stesso frame aggiungiamo un widget per l'inserimento.

```
Alias: txtPath
Text: FilePath
Width: 266
Height: 21
```

Aggiungiamo un pulsante alla destra del widget per l'inserimento.

```
Alias: btnSearchPath
Text: "..." (no quotes)
```

Aggiungiamo cinque (5) pulsanti di

selezione, da mettere nel seguente ordine...

```
x
x x
x x
```

I tre pulsanti per la selezione sulla sinistra sono per i file video e i due sulla destra sono per i file audio. Prima gestiremo i tre file sulla sinistra e e poi i due sulla destra.

```
Alias: chkAVI
Text: ".avi" (no quotes)
Variable: VchkAVI

Alias: chkMKV
Text: ".mkv" (no quotes)
Variable: VchkMKV

Alias: chkMV4
Text: ".mv4" (no quotes)
Variable: VchkMV4

Alias: chkMP3
Text: ".mp3" (no quotes)
Variable: VchkMP3

Alias: chkOGG
Text: ".ogg" (no quotes)
Variable: VchkOGG
```

Infine, aggiungiamo in questo frame un pulsante da qualche parte sotto i cinque pulsanti di selezione, centrandolo in qualche modo rispetto al frame.

```
Alias: btnGo
Text: GO!
```

Ora aggiungiamo un ulteriore frame sotto all'ultimo frame.

```
Width: 565
Height: 265
```

Ho posizionato il mio con X0 Y250. Potreste dover ridimensionare il vostro form principale per renderlo visibile interamente. All'interno di questo frame aggiungiamo un widget Scrolledtreeview.

```
Width: 550
Height: 254
X Position: 10
Y Position: 10
```

Ecco. Abbiamo disegnato la nostra GUI. Ora tutto quello che è rimasto da fare è di creare la lista funzioni e collegare le funzioni ai pulsanti.

Nella finestra con l'elenco delle funzioni, premere il pulsante New (quello più a sinistra). Questo farà apparire l'editor di nuove funzioni. Cambiamo il testo nella casella di inserimento Function da "py:xxx" a "py.btnExitClick()". Nella casella per l'inserimento degli argomenti, digitare "p1". Nella casella multi linea in basso, cambiare il testo in:

```
def btnExitClick(p1):

    sys.exit()
```

Notare che non è indentato. Page lo farà per noi quando creerà il file python.

Successivamente creiamo una altra funzione chiamata btnGoClick. Ricordarsi di aggiungere il parametro passato con "p1". Lasciare l'istruzione "pass". Si cambierà più tardi.

Infine, aggiungiamo una altra funzione chiamata "btnSearchPath". Di nuovo, lasciare l'istruzione pass.

Al termine, bisogna collegare i pulsanti alle funzioni appena create.

Fare clic con il tasto destro sul pulsante exit creato e selezionare Bind. Si aprirà una grande casella. Fare clic sul pulsante New bind, poi su Button-1 e cambiare la parola "TODO" nella casella di inserimento a destra con "btnExitClick". NON inserire le parentesi () qui.

Colleghiamo il pulsante GO a btnGoClick e il pulsante "..." a btnSearchPathClick.

Salvare la GUI e generare il codice python.

Ora ciò che resta è di creare il codice che "incolla" insieme la GUI.

Apriamo con l'editor preferito il codice appena generato. Cominciamo ad esaminare quello che Page ha creato per noi.

In cima al file c'è l'intestazione standard python e una singola istruzione di importazione per la libreria sys. Poi c'è del codice abbastanza confuso (a prima vista). Questo, essenzialmente, analizza la versione di python con cui si sta cercando di eseguire l'applicazione e quindi importa la corretta versione delle librerie tkinter. A meno che non si stia usando python 3.x, si possono fondamentalmente ignorare le ultimi due.

Modificheremo la porzione di codice 2.x per importare altri moduli tkinter tra poco.

La prossima è la routine "vp_start_gui()". È la routine principale del programma. Imposta la nostra GUI, le variabili di cui abbiamo bisogno e inoltre richiama il ciclo principale tkinter. Si noterà la linea "w = None" al di sotto di questa. Non è indentata e non si suppone che lo sia.

Le successive sono due routine (create_Searcher e destroy_Searcher)

che sono usate per sostituire il ciclo principale se si sta invocando questa applicazione come libreria. Non dobbiamo preoccuparci di queste.

Poi c'è la routine "set_Tk_var". Si definiscono le variabili tkinter utilizzate che necessitano di essere assegnate prima di creare i widget. È possibile riconoscerle come le variabili di testo per il widget di inserimento FilePath e le variabili per le caselle di selezione. Le successive tre routine sono quelle create usando l'editor di funzioni e una funzione di inizializzazione "init()".

Eseguiamo ora il programma. Notare che le caselle di selezione sono ingrigite e selezionate. Non è ciò che vogliamo per il rilascio dell'applicazione, così creeremo un po' di codice per pulirli prima che il form sia mostrato all'utente. L'unica altra cosa funzionante, oltre alle caselle di selezione, è il pulsante Exit.

Andiamo avanti e terminiamo il programma.

Ora diamo una occhiata alla classe che contiene veramente la definizione della GUI. Che sarebbe la "class Searcher". Qui è dove sono definiti tutti i widget e disposti nel nostro form. Dovreste ormai avere familiarità con questo.

Due ulteriori classi che contengono il codice per la vista ad albero scorrevole sono create per noi. Non dobbiamo cambiare niente in queste. E' stato creato tutto da Page per noi.

Ora torniamo all'inizio del codice e iniziamo le modifiche.

Abbiamo bisogno di importare pochi moduli di libreria in più, così sotto l'istruzione "import sys", aggiungiamo...

```
import os

from os.path import join,
getsize, exists
```

Ora troviamo la sezione che ha la linea "py2 = True". Come detto prima, è la sezione relativa alle importazioni di tkinter per Python versione 2.x. Sotto a "import ttk", bisogna aggiungere le seguenti istruzioni per supportare la libreria FileDialog. Serve anche importare il modulo tkFont.

```
import tkFileDialog

import tkFont
```

Poi bisogna aggiungere alcune variabili alla routine "set_Tk_var()". In fondo alla routine aggiungiamo le seguenti linee...

```
global exts, FileList

exts = []

FileList=[]
```

Qui creiamo due variabili gloabali (exts e FileList) che saranno accessibili in seguito nel nostro codice. Entrambe sono liste. "exts" è un lista delle estensioni che l'utente seleziona dalla GUI. "FileList" contiene una lista di liste dei file corrispondenti trovati quando noi facciamo la nostra ricerca. Useremo questa per popolare il widget della vista ad albero.

Poiché il nostro "btnExitClick" è già stato fatto per noi da Page, gestiremo la routine "btnGoClick". Commentiamo l'istruzione pass e aggiungiamo il codice in modo tale che somigli a questo…

```
def btnGoClick(p1) :

#pass

BuildExts()

fp = FilePath.get()

e1 = tuple(exts)

Walkit(fp,e1)

LoadDataGrid()
```

Questa è la routine che sarà chiamata quando l'utente preme il pulsante "GO!". Lancia una routine denominata "BuildExts" che crea la lista delle estensioni che l'utente ha selezionato. Si ottiene quindi il percorso che l'utente ha selezionato dalla finestra di dialogo AskDirectory, che viene assegnato alla variabile fp. Crea quindi una tupla (ndt elemento di una relazione con attributi) dalla lista delle estensioni, che è necessaria quando si controllano i file. Poi lancia una routine denominata "Walkit", passandogli la directory di destinazione e l'estensione tupla.

Infine lancia una routine chiamata "LoadDatagrid".

Successivamente si deve rimpolpare la routine "btnSearchPathClick". Commentiamo l'istruzione pass e cambiamo il codice in modo che somigli a questo...

```
def btnSearchPathClick(p1) :

      #pass

      path =
tkFileDialog.askdirectory()
#**self.file_opt)

      FilePath.set(path)
```

La prossima è la routine init. Di nuovo, facciamo in modo che il codice

somigli a questo...

```
def init():

      #pass

      # Fires AFTER Widgets and
Window are created...

      global treeview

      BlankChecks()

      treeview =
w.Scrolledtreeview1

      SetupTreeview()
```

Questa crea una variabile globale chiamata "treeview". Poi lancia una routine che pulirà le selezioni grigie dalle caselle di selezione, designa la variabile "treeview" a puntare alla vista ad albero scorrevole nel nostro form e lancia "SetupTreeview" per impostare le intestazioni delle colonne.

Qui c'è il codice per la routine "BlanckChecks" che deve essere la prossima.

```
def BlankChecks():

      VchkAVI.set('0')

      VchkMKV.set('0')

      VchkMP3.set('0')

      VchkMV4.set('0')

      VchkOGG.set('0')
```

Qua, tutto ciò che stiamo facendo è impostare le variabili (che impostano automaticamente lo stato di selezione delle caselle di selezione) a "0". Se ricordate, ogni volta che la casella di selezione è premuta, la variabile è automaticamente aggiornata. Se la variabile è cambiata dal nostro codice, la casella di selezione risponde altrettanto bene. Ora (in alto a destra) gestiremo la routine che costruisce la lista delle estensioni in base a ciò che l'utente ha premuto.

Tornate indietro con la memoria al mio nono articolo in FCM#35. Scrissi del codice per creare un catalogo di file MP3. Useremo una versione accorciata di quella routine (al centro destra). Fate riferimento a FCM#35 se avete domande su questa routine.

Successivamente (in basso a destra) lanciamo la routine SetupTreeview. E' abbastanza semplice. Definisce una variabile "ColHeads" con le intestazioni che vogliamo in ciascuna colonna della vista ad albero. Lo fa tipo una lista. Assegna quindi l'attributo intestazione per ciascuna colonna. Assegna anche la larghezza della colonna alla dimensione di questa intestazione.

Infine dobbiamo creare la routine "LoadDataGrid (prossima pagina in alto a destra), dove caricare i dati nella vista ad albero. Ciascuna riga della vista ad albero è una voce nella variabile della lista FileList. Regoliamo inoltre la larghezza di ciascuna colonna (di nuovo) per corrispondere alla dimensione della colonna dei dati.

Questa è la prima vista della applicazione. Avviamola e vediamo come l'abbiamo fatta. Notare che se si hanno tanti file da sfogliare, il programma sembra non rispondere. Questo è qualcosa che deve essere risolto. Creeremo una routine per

```
def BuildExts():
     if VchkAVI.get() == '1':
          exts.append(".avi")
     if VchkMKV.get() == '1':
          exts.append(".mkv")
     if VchkMP3.get() == '1':
          exts.append(".mp3")
     if VchkMV4.get() == '1':
          exts.append(".mv4")
     if VchkOGG.get() == '1':
          exts.append(".ogg")
```

```
def Walkit(musicpath,extensions):
     rcntr = 0
     fl = []
     for root, dirs, files in os.walk(musicpath):
          rcntr += 1 # This is the number of folders we have walked
          for file in [f for f in files if f.endswith(extensions)]:
               fl.append(file)
               fl.append(root)
               FileList.append(fl)
               fl=[]
```

```
def SetupTreeview():
     global ColHeads
     ColHeads = ['Filename','Path']
     treeview.configure(columns=ColHeads,show="headings")
     for col in ColHeads:
          treeview.heading(col, text = col.title(),
               command = lambda c = col: sortby(treeview, c, 0))
     ## adjust the column's width to the header string
     treeview.column(col, width =
tkFont.Font().measure(col.title()))
```

cambiare il cursore predefinito con uno in stile "orologio" e viceversa per le operazioni che richiedono molto tempo, così l'utente lo noterà.

Nella routine "set_Tk_var", aggiungeremo il seguente codice in fondo.

```
global busyCursor,preBusyCursors,busyWidgets

busyCursor = 'watch'

preBusyCursors = None

busyWidgets = (root, )
```

Quello che facciamo qui è definire le variabili globali, inizializzarle e impostare quindi il widget (in busyWidgets) che vogliamo che risponda ai cambiamenti del cursore. In questo caso lo impostiamo a root che è la nostra finestra completa. Notare che questa è un tupla.

Poi creiamo due routine per impostare e per ripristinare il cursore. Prima la routine di impostazione, che chiameremo "busyStart". Dopo la routine "loadDataGrid", inseriamo il codice mostrato a centro destra.

Dobbiamo prima verificare se un valore è stato passato a "newcursor". In caso contrario, si preimposta a busyCursor. Poi passiamo attraverso la tupla busyWidgets e impostiamo il cursore a qualsiasi cosa vogliamo.

Ora inseriamo il codice mostrato in basso a destra.

In questa routine, azzeriamo semplicemente il cursore per i widget nella nostra tupla busyWidget riportandolo alla forma predefinita.

Salviamo ed eseguiamo il programma. Dovremmo notare cambiamenti del cursore ogni volta che si scorre una lunga lista di file.

Sebbene questa applicazione non fa molto, mostra come usare Page per creare un ambiente di sviluppo veramente veloce. Dall'articolo di oggi possiamo vedere come, avendo in anticipo una buona progettazione della GUI, si può rendere il processo di sviluppo facile e abbastanza indolore.

Il file tcl è salvato in pastebin presso http://pastebin.com/AA1kE4Dy e il codice python presso http://pastebin.com/VZm5un3e.

Ci vediamo la prossima volta.

```
def LoadDataGrid():
     global ColHeads
     for c in FileList:
          treeview.insert('','end',values=c)
          # adjust column's width if necessary to fit each value
          for ix, val in enumerate(c):
               col_w = tkFont.Font().measure(val)
               if treeview.column(ColHeads[ix],width=None)<col_w:
                    treeview.column(ColHeads[ix], width=col_w)
```

```
def busyStart(newcursor=None):
     global preBusyCursors
     if not newcursor:
          newcursor = busyCursor
     newPreBusyCursors = {}
     for component in busyWidgets:
          newPreBusyCursors[component] = component['cursor']
          component.configure(cursor=newcursor)
          component.update_idletasks()
          preBusyCursors = (newPreBusyCursors, preBusyCursors)
```

```
def busyEnd():
     global preBusyCursors
     if not preBusyCursors:
          return
     oldPreBusyCursors = preBusyCursors[0]
     preBusyCursors = preBusyCursors[1]
     for component in busyWidgets:
          try:
               component.configure(cursor=oldPreBusyCursors[component])
          except KeyError:
               pass
          component.update_idletasks()
```