

Burkhard A. Meier

Python GUI Programming Cookbook

Second Edition

Develop beautiful and powerful GUIs using the Python programming language



Packt

Programmazione GUI Python

Libro di cucina

Seconda edizione

Sviluppa GUI belle e potenti usando il linguaggio di programmazione Python

Burkhard A. Meier

Packt

BIRMINGHAM - MUMBAI

Ricettario per la programmazione della GUI Python

Seconda edizione

Copyright © 2017 Packt Publishing

Tutti i diritti riservati. Nessuna parte di questo libro può essere riprodotta, archiviata in un sistema di recupero o trasmessa in qualsiasi forma o con qualsiasi mezzo, senza previa autorizzazione scritta dell'editore, tranne nel caso di brevi citazioni incorporate in articoli critici o recensioni.

Ogni sforzo è stato fatto nella preparazione di questo libro per garantire l'accuratezza delle informazioni presentate. Tuttavia, le informazioni contenute in questo libro sono vendute senza garanzia, esplicita o implicita. Né l'autore, né Packt Publishing, ei suoi rivenditori e distributori saranno ritenuti responsabili per eventuali danni causati o presumibilmente causati direttamente o indirettamente da questo libro.

Packt Publishing si è sforzata di fornire informazioni sui marchi su tutte le società e i prodotti menzionati in questo libro mediante l'uso appropriato delle maiuscole.

Tuttavia, Packt Publishing non può garantire l'accuratezza di queste informazioni.

Prima pubblicazione: novembre 2015

Seconda edizione: maggio 2017

Riferimento di produzione: 1190517

Pubblicato da Packt Publishing Ltd.

Livery Place

35 Livrea Street

Birmingham

B3 2PB, Regno Unito.

ISBN 978-1-78712-945-0

www.packtpub.com

Get
80%
off any Packt tech eBook or Video!



Go to www.packtpub.com
and use this code in the
checkout:

HBMAPT80OFF

Packt

Titoli di coda

Autore

Burkhard A. Meier

Editor di copia

Muktikant Garimella

Recensore

Mohit

Coordinatore del progetto

Ulhas Kambali

Commissioning Editor

Kunal Parikh

Correttore di bozze

Modifica Safis

Editor delle acquisizioni

Denim Pinto

Indicizzatore

Aishwarya Gangawane

Editor per lo sviluppo dei contenuti

Anurag Ghogre

Grafica

Abhinash Sahu

Redattore tecnico

Prashant Mishra

Coordinatore di produzione

Nilesh Mohite

Circa l'autore

Burkhard A. Meier ha più di 17 anni di esperienza professionale nel settore del software come tester e sviluppatore di software, specializzato nello sviluppo, nell'esecuzione e nell'analisi dell'automazione dei test del software. Ha una solida esperienza nello sviluppo dell'automazione dei test del software Python 3, nonché nell'amministrazione di database relazionali SQL, nello sviluppo di stored procedure e nel debug del codice.

Pur avendo esperienza in Visual Studio .NET C#, Visual Test, TestComplete e altri linguaggi di test (come C/C++), l'obiettivo principale dell'autore negli ultimi cinque anni è stato lo sviluppo dell'automazione dei test scritta in Python 3 per testare i principali edge delle termocamere FLIR ONE (ora alla terza generazione) per smartphone iPhone e Android e tablet palmari, oltre a garantire la qualità delle piattaforme di termocamere FLIR bolometer IR.

Appassionato di arte, bellezza e programmazione, l'autore ha sviluppato GUI in C# e Python per semplificare le attività quotidiane di automazione dei test, consentendo a questi test automatizzati di essere eseguiti incustoditi per settimane, raccogliendo dati molto utili da analizzare, tracciare automaticamente in grafici e inviato via e-mail alla direzione superiore al completamento delle esecuzioni di test automatizzate notturne.

I suoi precedenti lavori includono il lavoro come ingegnere e progettista senior dell'automazione dei test per InfoGenesis (ora Agilysys), QAD, InTouch Health e FLIR Systems.

Puoi metterti in contatto con lui tramite il suo account LinkedIn, <https://www.linkedin.com/pub/burkhard-meier/5/246/296>.

Vorrei ringraziare tutti gli artisti veramente grandi, come Leonardo da Vinci, Charles Baudelaire, Edgar Allan Poe e tanti altri per aver portato la presenza della bellezza nelle nostre vite umane. Questo libro riguarda la creazione di bellissime GUI scritte nel linguaggio di programmazione Python ed è stato ispirato da questi artisti davvero grandiosi.
Vorrei ringraziare tutte le persone fantastiche che hanno reso possibile questo libro. Senza nessuno di voi, questo libro esisterebbe solo nella mia mente. Vorrei ringraziare in particolare tutti i miei editori di Packt Publishing: Sonali, Anurag, Prashant, Vivek, Arwa, Sumeet, Saurabh, Pramod, Nikhil e molti altri. Vorrei anche ringraziare tutti i revisori del codice di questo libro. Senza di loro, questo libro sarebbe più difficile da leggere e applicare ai problemi del mondo reale. Ultimo ma non meno importante, vorrei ringraziare mia moglie, nostra figlia e i nostri genitori per il supporto emotivo che hanno fornito con tanto successo durante la stesura della seconda edizione di questo libro. Vorrei anche ringraziare il creatore del bellissimo e potente linguaggio di programmazione che Python è veramente. Grazie Guido.

Informazioni sul revisore

Mohit (mohitraj.cs@gmail.com) è un programmatore Python con un vivo interesse nel campo della sicurezza delle informazioni. Ha completato la sua laurea in tecnologia in informatica presso la Kurukshetra University, Kurukshetra, e il master in ingegneria (2012) in informatica presso la Thapar University, Patiala. È un C|EH, ECSA dell'EC-Council USA ed ex IBMer. Ha pubblicato diversi articoli su riviste nazionali e internazionali. È l'autore di Python Penetration Testing Essentials e Python Penetration Testing for Developers, sempre di Packt Publishing.

www.PacktPub.com

Per file di supporto e download relativi al tuo libro, visita, www.PacktPub.com.

Sapevi che Packt offre versioni eBook di ogni libro pubblicato, con file PDF ed ePub disponibili? Puoi aggiornare alla versione eBook su www.PacktPub.com e come cliente di un libro cartaceo, hai diritto a uno sconto sulla copia dell'eBook. Mettiti in contatto con noi a service@packtpub.com per ulteriori dettagli.

A www.PacktPub.com, puoi anche leggere una raccolta di articoli tecnici gratuiti, iscriverti a una serie di newsletter gratuite e ricevere sconti e offerte esclusivi su libri ed eBook Packt.



<https://www.packtpub.com/mapt>

Ottieni le competenze software più richieste con Mapt. Mapt ti dà pieno accesso a tutti i libri Packt e ai corsi video, oltre a strumenti leader del settore per aiutarti a pianificare il tuo sviluppo personale e far avanzare la tua carriera.

Perché abbonarsi?

- Completamente ricercabile in tutti i libri pubblicati da Packt Copia e
- incolla, stampa e aggiungi contenuti ai segnalibri Su richiesta e
- accessibili tramite un browser web

Opinione del cliente

Grazie per aver acquistato questo libro Packt. In Packt, la qualità è al centro del nostro processo editoriale. Per aiutarci a migliorare, lasciaci una recensione onesta sulla pagina Amazon di questo libro a <https://www.amazon.com/dp/1787129454>.

Se desideri unirti al nostro team di revisori regolari, puoi inviarci un'e-mail a customerreviews@packtpub.com. Premiamo i nostri revisori regolari con eBook e video gratuiti in cambio del loro prezioso feedback. Aiutaci a essere implacabili nel migliorare i nostri prodotti!

Sommario

Prefazione	1
Capitolo 1: Creazione del modulo GUI e aggiunta di widget	7
introduzione	7
Creazione della nostra prima GUI Python	9
Prepararsi	9
Come farlo...	10
Come funziona...	10
C'è più...	11
Prevenire il ridimensionamento della GUI	12
Prepararsi	12
Come farlo...	12
Come funziona...	13
Aggiunta di un'etichetta al modulo GUI	14
Prepararsi	14
Come farlo...	14
Come funziona...	15
C'è più...	16
Creazione di pulsanti e modifica delle loro proprietà di testo	16
Prepararsi	17
Come farlo...	17
Come funziona...	18
C'è più...	18
Widget casella di testo	19
Prepararsi	19
Come farlo...	19
Come funziona...	20
Impostare lo stato attivo su un widget e disabilitare i widget	21
Prepararsi	21
Come farlo...	21
Come funziona...	23
C'è più...	23
Widget della casella combinata	24
Prepararsi	24
Come farlo...	24

Come funziona...	25
C'è più...	26
Creazione di un pulsante di spunta con diversi stati iniziali	26
Prepararsi	26
Come farlo...	27
Come funziona...	28
Utilizzo dei widget dei pulsanti di opzione	28
Prepararsi	29
Come farlo...	29
Come funziona...	30
C'è più...	31
Utilizzo di widget di testo scorrevole	31
Prepararsi	31
Come farlo...	32
Come funziona...	33
Aggiunta di più widget in un ciclo	34
Prepararsi	34
Come farlo...	34
Come funziona...	35
C'è più...	35
Capitolo 2: Gestione del layout	36
introduzione	36
Disposizione di più etichette all'interno di un widget cornice etichetta	38
Prepararsi	38
Come farlo...	38
Come funziona...	40
C'è più...	41
Utilizzo della spaziatura interna per aggiungere spazio intorno ai widget	41
Prepararsi	41
Come farlo...	41
Come funziona...	42
Come i widget espandono dinamicamente la GUI	44
Prepararsi	45
Come farlo...	45
Come funziona...	49
C'è più...	49
Allineare i widget della GUI incorporando i frame all'interno dei frame	49
Prepararsi	49
Come farlo...	50

Come funziona...	53
Creazione di barre dei menu	54
Prepararsi	55
Come farlo...	55
Come funziona...	61
C'è più...	62
Creazione di widget a schede	62
Prepararsi	62
Come farlo...	63
Come funziona...	68
Utilizzo del gestore del layout della griglia	68
Prepararsi...	68
Come farlo...	68
Come funziona...	70
Capitolo 3: Personalizzazione dell'aspetto e della sensazione	71
introduzione	71
Creazione di finestre di messaggio: informazioni, avvisi ed errori	72
Prepararsi	73
Come farlo...	73
Come funziona...	75
Come creare finestre di messaggio indipendenti independent	77
Prepararsi	77
Come farlo...	77
Come funziona...	80
Come creare il titolo di un modulo finestra tkinter	81
Prepararsi	81
Come farlo...	81
Come funziona...	81
Cambiare l'icona della finestra principale di root	82
Prepararsi	82
Come farlo...	82
Come funziona...	83
Utilizzo di un controllo della casella numerica	83
Prepararsi	83
Come farlo...	83
Come funziona...	87
Rilievo, aspetto incavato e rialzato dei widget	87
Prepararsi	87
Come farlo...	88

Come funziona...	89
Creazione di tooltip usando Python	90
Prepararsi	90
Come farlo...	91
Come funziona...	93
Aggiunta di una barra di avanzamento alla GUI	94
Prepararsi	94
Come farlo...	95
Come funziona...	97
Come usare il widget della tela	97
Prepararsi	97
Come farlo...	98
Come funziona...	98
Capitolo 4: Dati e classi	100
introduzione	100
Come usare StringVar()	102
Prepararsi	102
Come farlo...	103
Come funziona...	105
Come ottenere dati da un widget	108
Prepararsi	108
Come farlo...	108
Come funziona...	109
Utilizzo di variabili globali a livello di modulo	110
Prepararsi	110
Come farlo...	110
Come funziona...	111
In che modo la codifica nelle classi può migliorare la GUI	114
Prepararsi	115
Come farlo...	115
Come funziona...	120
Scrivere funzioni di callback	120
Prepararsi	121
Come farlo...	121
Come funziona...	121
Creazione di componenti GUI riutilizzabili	122
Prepararsi	122
Come farlo...	122
Come funziona...	126

Capitolo 5: Grafici Matplotlib	127
introduzione	127
Creare bellissimi grafici usando Matplotlib	128
Prepararsi	128
Come farlo...	129
Come funziona...	131
Installazione di Matplotlib usando pip con estensione whl	131
Prepararsi	131
Come farlo...	134
Come funziona...	137
Creare il nostro primo grafico	138
Prepararsi	138
Come farlo...	138
Come funziona...	139
Posizionamento di etichette sui grafici	140
Prepararsi	140
Come farlo...	140
Come funziona...	145
Come dare al grafico una legenda	146
Prepararsi	146
Come farlo...	146
Come funziona...	149
Grafici di ridimensionamento	149
Prepararsi	150
Come farlo...	150
Come funziona...	151
Regolazione dinamica della scala dei grafici	152
Prepararsi	152
Come farlo...	152
Come funziona...	156
Capitolo 6: Thread e networking	157
introduzione	157
Come creare più thread	159
Prepararsi	160
Come farlo...	160
Come funziona...	163
Avvio di un thread	163
Prepararsi	163

Come farlo...	165
Come funziona...	168
Interrompere un thread	169
Prepararsi	169
Come farlo...	169
Come funziona...	172
Come usare le code	173
Prepararsi	173
Come farlo...	174
Come funziona...	179
Passaggio di code tra moduli diversi	179
Prepararsi	180
Come farlo...	180
Come funziona...	182
Utilizzo dei widget di dialogo per copiare i file nella rete	183
Prepararsi	183
Come farlo...	183
Come funziona...	193
Utilizzo di TCP/IP per comunicare in rete	194
Prepararsi	194
Come farlo...	194
Come funziona...	197
Utilizzo di urlopen per leggere i dati dai siti Web	197
Prepararsi	197
Come farlo...	197
Come funziona...	201
Capitolo 7: Memorizzazione dei dati nel nostro database MySQL tramite la nostra GUI	202
introduzione	202
Installazione e connessione a un server MySQL da Python	204
Prepararsi	204
Come farlo...	207
Come funziona...	210
Configurazione della connessione al database MySQL	210
Prepararsi	211
Come farlo...	211
Come funziona...	214
Progettare il database della GUI Python	215
Prepararsi	215
Come farlo...	215

Come funziona...	222
Utilizzo del comando SQL INSERT	222
Prepararsi	223
Come farlo...	223
Come funziona...	225
Utilizzo del comando SQL UPDATE	225
Prepararsi	226
Come farlo...	226
Come funziona...	230
Utilizzo del comando SQL DELETE	231
Prepararsi	231
Come farlo...	231
Come funziona...	235
Memorizzazione e recupero dei dati dal nostro database MySQL	235
Prepararsi	235
Come farlo...	235
Come funziona...	239
Utilizzo del workbench MySQL	239
Prepararsi	240
Come farlo...	240
Come funziona...	246
C'è più...	246
Capitolo 8: Internazionalizzazione e test	247
introduzione	247
Visualizzazione del testo del widget in diverse lingue	249
Prepararsi	249
Come farlo...	249
Come funziona...	251
Cambiare l'intera lingua della GUI, tutto in una volta	252
Prepararsi	252
Come farlo...	252
Come funziona...	257
Localizzare la GUI	257
Prepararsi	258
Come farlo...	258
Come funziona...	262
Preparazione della GUI per l'internazionalizzazione	263
Prepararsi	263
Come farlo...	263

Come funziona...	267
Come progettare una GUI in modo agile	267
Prepararsi	268
Come farlo...	268
Come funziona...	271
Dobbiamo testare il codice della GUI?	271
Prepararsi	272
Come farlo...	272
Come funziona...	275
Impostazione degli orologi di debug	275
Prepararsi	276
Come farlo...	276
Come funziona...	280
Configurazione di diversi livelli di output di debug	280
Prepararsi	280
Come farlo...	281
Come funziona...	283
Creazione di codice di autotest utilizzando la sezione <code>_main_</code> di Python	284
Prepararsi	284
Come farlo...	284
Come funziona...	289
Creazione di GUI robuste utilizzando i test di unità	289
Prepararsi	289
Come farlo...	289
Come funziona...	293
Come scrivere unit test utilizzando l'IDE Eclipse PyDev	293
Prepararsi	294
Come farlo...	294
Come funziona...	300
Capitolo 9: Estendere la nostra GUI con la libreria wxPython	301
introduzione	301
Installazione della libreria wxPython	303
Prepararsi	303
Come farlo...	303
Come funziona...	306
Creazione della nostra GUI in wxPython	306
Prepararsi	307
Come farlo...	307
Come funziona...	311

Aggiunta rapida di controlli utilizzando wxPython	312
Prepararsi	312
Come farlo...	312
Come funziona...	317
Cercando di incorporare un'app wxPython principale in un'app tkinter principale	318
Prepararsi	318
Come farlo...	319
Come funziona...	320
Cercando di incorporare il nostro codice GUI di tkinter in wxPython	321
Prepararsi	321
Come farlo...	321
Come funziona...	323
Utilizzo di Python per controllare due diversi framework GUI GUI	324
Prepararsi	324
Come farlo...	324
Come funziona...	326
Comunicazione tra le due GUI collegate	327
Prepararsi	328
Come farlo...	328
Come funziona...	332
Capitolo 10: Creazione di fantastiche GUI 3D con PyOpenGL e PyGLet	333
introduzione	333
PyOpenGL trasforma la nostra GUI	335
Prepararsi	335
Come farlo...	336
Come funziona...	339
La nostra GUI in 3D!	339
Prepararsi	340
Come farlo...	340
Come funziona...	344
Usare le bitmap per rendere carina la nostra GUI	345
Prepararsi	345
Come farlo...	346
Come funziona...	348
PyGLet trasforma la nostra GUI più facilmente di PyOpenGL	348
Come farlo...	349
Come funziona...	351
La nostra GUI in fantastici colori	351
Prepararsi	352

Come farlo...	352
Come funziona...	355
Animazione OpenGL	355
Prepararsi	356
Come farlo...	356
Come funziona...	362
Creare una presentazione usando tkinter	362
Prepararsi	363
Come farlo...	363
Come funziona...	368
Capitolo 11: Migliori Pratiche	369
introduzione	369
Evitare il codice spaghetti	370
Prepararsi	371
Come farlo...	371
Come funziona...	374
Usare __init__ per connettere i moduli	377
Prepararsi	378
Come farlo...	378
Come funziona...	383
Miscelazione fall-down e codifica OOP	384
Prepararsi	384
Come farlo...	384
Come funziona...	388
Utilizzo di una convenzione di denominazione in codice	388
Prepararsi	389
Come farlo...	389
Come funziona...	391
Quando non usare OOP	392
Prepararsi	392
Come farlo...	392
Come funziona...	396
Come utilizzare con successo i modelli di progettazione	396
Prepararsi	396
Come farlo...	396
Come funziona...	399
Evitare la complessità	399
Prepararsi	399
Come farlo...	400

Come funziona...	404
Progettazione della GUI utilizzando più notebook	
Prepararsi	405
Come farlo...	405
Come funziona...	409
Indice	413

Prefazione

Nella seconda edizione di questo libro, esploreremo il meraviglioso mondo delle interfacce utente grafiche (GUI) utilizzando il linguaggio di programmazione Python. Useremo l'ultima versione di Python 3. Tutte le ricette della prima edizione sono incluse in questa edizione. Abbiamo aggiunto alcune nuove ricette alla Seconda Edizione, che potresti non trovare facilmente tramite una ricerca su Google. Penso che queste nuove ricette saranno utili e interessanti per il lettore.

Questo è un ricettario di programmazione. Ogni capitolo è autonomo e spiega una determinata soluzione di programmazione. Inizieremo in modo molto semplice, ma nel corso di questo libro creeremo un'applicazione funzionante scritta in Python 3. Ogni ricetta estenderà la creazione di questa applicazione. Lungo la strada, parleremo di reti, code, database, libreria grafica OpenGL e molte altre tecnologie. Applicheremo modelli di progettazione e utilizzeremo le migliori pratiche.

Il libro presuppone che il lettore abbia una certa esperienza nell'uso del linguaggio di programmazione Python, ma ciò non è realmente necessario per utilizzare con successo questo libro. Questo libro può anche essere usato come introduzione al linguaggio di programmazione Python, se e solo se ti dedichi al tuo desiderio di diventare un programmatore Python.

Se sei uno sviluppatore esperto in qualsiasi altra lingua, ti divertirai ad ampliare la tua cassetta degli attrezzi professionale aggiungendo GUI di scrittura utilizzando Python alla tua cassetta degli attrezzi. Siete pronti?

Iniziamo il nostro viaggio...

Di cosa tratta questo libro

Capitolo 1, *Creazione del modulo GUI e aggiunta di widget*, spiega come sviluppare la nostra prima GUI in Python. Inizieremo con il codice minimo richiesto per creare un'applicazione GUI in esecuzione. Ogni ricetta aggiunge quindi diversi widget al modulo GUI.

Capitolo 2, *Gestione del layout*, esplora come organizzare i widget per creare la nostra GUI Python. Il gestore del layout della griglia è uno degli strumenti di layout più importanti integrati in tkinter che utilizzeremo.

Capitolo 3, *Look and Feel personalizzazione*, mostra diversi esempi di come creare una buona interfaccia grafica. A livello pratico, aggiungeremo funzionalità alla Guida | A proposito di voce di menu che abbiamo creato in una delle ricette.

Capitolo 4, *Dati e Classi*, discute il salvataggio dei dati visualizzati dalla nostra GUI. Inizieremo a utilizzare la programmazione orientata agli oggetti (OOP) per estendere le funzionalità integrate di Python.

Capitolo 5, *Grafici Matplotlib*, spiega come creare bellissimi grafici che rappresentano visivamente i dati. A seconda del formato dell'origine dati, possiamo tracciare una o più colonne di dati all'interno dello stesso grafico.

Capitolo 6, *Thread e reti*, spiega come estendere le funzionalità della nostra GUI Python utilizzando thread, code e connessioni di rete. Questo ci mostrerà che la nostra GUI non è affatto limitata all'ambito locale del nostro PC.

Capitolo 7, *Memorizzazione dei dati nel nostro database MySQL tramite la nostra GUI*, ci mostra come connetterci a un server di database MySQL. La prima ricetta in questo capitolo mostrerà come installare la MySQL Server Community Edition gratuita, e nelle seguenti ricette creeremo database, tabelle, quindi caricheremo i dati in quelle tabelle e modificheremo questi dati.

Leggeremo anche i dati dal server MySQL nella nostra GUI.

Capitolo 8, *Internazionalizzazione e Sperimentazione*, mostra come internazionalizzare la nostra GUI visualizzando il testo su etichette, pulsanti, schede e altri widget in diverse lingue. Inizieremo in modo semplice e poi esploreremo come possiamo preparare la nostra GUI per l'internazionalizzazione a livello di progettazione. Esploreremo anche diversi modi per testare automaticamente la nostra GUI utilizzando il framework di test delle unità integrato di Python.

Capitolo 9, *Estendere la nostra GUI con la libreria wxPython*, introduce un altro toolkit GUI Python che attualmente non viene fornito con Python. Si chiama wxPython e useremo la versione Phoenix di wxPython, che è stata progettata per funzionare bene con Python 3.

Capitolo 10, *Creazione di fantastiche GUI 3D con PyOpenGL e PyGlet*, mostra come trasformare la nostra GUI dandogli vere capacità tridimensionali. Useremo due pacchetti Python di terze parti. PyOpenGL è un'associazione Python allo standard OpenGL, che è una libreria grafica integrata in tutti i principali sistemi operativi. Ciò conferisce ai widget risultanti un aspetto e una sensazione nativi. Pyglet è un altro di questi collegamenti che esploreremo in questo capitolo. Mostreremo anche del codice che utilizza direttamente la libreria PyOpenGL. Questo è un approccio di basso livello che potrebbe aprire alcune porte al lettore interessato.

Capitolo 11, *Migliori pratiche*, esplora diverse best practice che possono aiutarci a costruire la nostra GUI in modo efficiente e mantenerla sia gestibile che estendibile. Le migliori pratiche sono applicabili a qualsiasi buon codice e la nostra GUI non fa eccezione alla progettazione e all'implementazione di buone pratiche software.

Cosa ti serve per questo libro

Tutto il software necessario per questo libro è disponibile online ed è gratuito. Questo inizia con Python 3 stesso e poi si estende ai moduli aggiuntivi di Python. Per scaricare qualsiasi software richiesto, avrai bisogno di una connessione Internet funzionante.

A chi è rivolto questo libro

Questo libro è per i programmati che desiderano creare una GUI. Potresti essere sorpreso da ciò che possiamo ottenere creando GUI belle, funzionali e potenti utilizzando il linguaggio di programmazione Python. Python è un linguaggio di programmazione meraviglioso e intuitivo ed è molto facile da imparare.

Ti invito a iniziare questo viaggio ora. Sarà molto divertente!

Convegni

In questo libro troverai una serie di stili di testo che distinguono tra diversi tipi di informazioni. Ecco alcuni esempi di questi stili e una spiegazione del loro significato.

Le parole in codice nel testo, i nomi delle tabelle del database, i nomi delle cartelle, i nomi dei file, le estensioni dei file, i nomi dei percorsi e l'input dell'utente sono mostrati come segue: "Utilizzando Python, possiamo creare le nostre classi utilizzando la classe **parola chiave** invece di **def parola chiave**."

Un blocco di codice è impostato come segue:

```
importa tkinter come tk
win = tk.Tk()
win.title("GUI Python")
win.mainloop()
```

Qualsiasi input o output della riga di comando viene scritto come segue:

```
pip install numpy-1.9.2+mkl-cp36-none-win_amd64.whl
```

I nuovi termini e le parole importanti sono mostrati in grassetto. Le parole che vedi sullo schermo, ad esempio nei menu o nelle finestre di dialogo, appaiono nel testo in questo modo: "Poi, aggiungeremo funzionalità alle voci di menu, ad esempio chiudendo la finestra principale facendo clic sul pulsante **Uscita** voce di menu e visualizzazione di a **Aiuto | Didi**logo."



Avvisi o note importanti vengono visualizzati in una casella come questa.



Suggerimenti e trucchi appaiono in questo modo.

Feedback dei lettori

Il feedback dei nostri lettori è sempre il benvenuto. Fateci sapere cosa ne pensate di questo libro, cosa vi è piaciuto o non vi è piaciuto. Il feedback dei lettori è importante per noi in quanto ci aiuta a sviluppare titoli da cui otterrai davvero il massimo.

Per inviarci un feedback generale, è sufficiente inviare un'e-mail feedback@packtpub.com, e menziona il titolo del libro nell'oggetto del tuo messaggio.

Se c'è un argomento in cui hai esperienza e sei interessato a scrivere o a contribuire a un libro, consulta la nostra guida all'autore all'indirizzo www.packtpub.com/authors.

Servizio Clienti

Ora che sei l'orgoglioso proprietario di un libro Packt, abbiamo una serie di cose per aiutarti a ottenere il massimo dal tuo acquisto.

Download del codice di esempio

Puoi scaricare i file di codice di esempio per questo libro dal tuo account su <http://www.packtpub.com>. Se hai acquistato questo libro altrove, puoi visitare <http://www.packtpub.com/support> e registrati per ricevere direttamente i file via e-mail.

Puoi scaricare i file di codice seguendo questi passaggi:

1. Accedi o registrati al nostro sito web utilizzando il tuo indirizzo e-mail e la password.
2. Passa il puntatore del mouse sul **SUPPORTO** scheda in alto.
3. Fare clic su **Download di codici ed errori**.
4. Inserisci il nome del libro nel campo **Ricerca** scatola.
5. Seleziona il libro per il quale stai cercando di scaricare i file di codice.

6. Scegli dal menu a discesa da dove hai acquistato questo libro.
7. Fare clic su **Scarica il codice**.

Una volta scaricato il file, assicurati di decomprimere o estrarre la cartella utilizzando l'ultima versione di:

- WinRAR / 7-Zip per Windows
- Zipeg / iZip / UnRarX per Mac 7-
- Zip / PeaZip per Linux

Anche il pacchetto di codici per il libro è ospitato su GitHub all'indirizzo <https://github.com/PacktPublishing/Python-GUI-Programmazione-Cookbook-Seconda-Edition>. Abbiamo anche un altro codice bundle dal nostro ricco catalogo di libri e video disponibili su <https://github.com/Packt Publishing/>. Dai un'occhiata!

Download delle immagini a colori di questo libro

Ti forniamo anche un file PDF con immagini a colori degli screenshot/diagrammi utilizzati in questo libro. Le immagini a colori ti aiuteranno a capire meglio i cambiamenti nell'output. Puoi scaricare questo file da https://www.packtpub.com/sites/default/files/downloads/PythonGUIProgrammingCookbookSecondEdition_ColorImages.pdf.

Errata

Sebbene abbiano preso ogni cura per garantire l'accuratezza dei nostri contenuti, gli errori si verificano. Se trovi un errore in uno dei nostri libri, forse un errore nel testo o nel codice, ti saremmo grati se potessi segnalarcelo. In questo modo, puoi salvare altri lettori dalla frustrazione e aiutarci a migliorare le versioni successive di questo libro. Se trovi qualche errata, ti preghiamo di segnalarcela visitando <http://www.packtpub.com/submit-errata>, selezionando il tuo libro, cliccando su **Modulo di invio errata** link e inserendo i dettagli della tua errata. Una volta verificata la tua errata, la tua richiesta sarà accettata e l'errata verrà caricata sul nostro sito Web o aggiunta a qualsiasi elenco di errata esistenti nella sezione Errata di quel titolo.

Per visualizzare gli errata inviati in precedenza, vai su <https://www.packtpub.com/books/content/support> inserisci il nome del libro nel campo di ricerca. Le informazioni richieste appariranno sotto il **Errata** sezione.

Pirateria

La pirateria di materiale protetto da copyright su Internet è un problema costante in tutti i media. In Packt, prendiamo molto sul serio la protezione del nostro copyright e delle nostre licenze. Se ti imbatti in copie illegali delle nostre opere in qualsiasi forma su Internet, ti preghiamo di fornirci immediatamente l'indirizzo della posizione o il nome del sito Web in modo che possiamo perseguiro un rimedio.

Si prega di contattarci a copyright@packtpub.com con un collegamento al materiale sospetto piratato.

Apprezziamo il tuo aiuto nel proteggere i nostri autori e la nostra capacità di offrirti contenuti di valore.

Domande

Se hai un problema con qualsiasi aspetto di questo libro, puoi contattarci all'indirizzo domande@packtpub.com, e faremo del nostro meglio per affrontare il problema.

1

Creazione del modulo GUI e Aggiunta di widget

In questo capitolo, iniziamo a creare fantastiche GUI utilizzando Python 3.6 e versioni successive. Tratteremo i seguenti argomenti:

- Creazione della nostra prima GUI Python
- Prevenzione del ridimensionamento della GUI
- Aggiunta di un'etichetta al modulo GUI
- Creazione di pulsanti e modifica delle loro proprietà di testo
- Widget casella di testo
- Impostare lo stato attivo su un widget e disabilitare i widget
- Widget casella combinata
- Creazione di un pulsante di spunta con diversi stati iniziali Utilizzo
- dei widget dei pulsanti di opzione
- Utilizzo di widget di testo scorrevole
- Aggiunta di più widget in un ciclo

introduzione

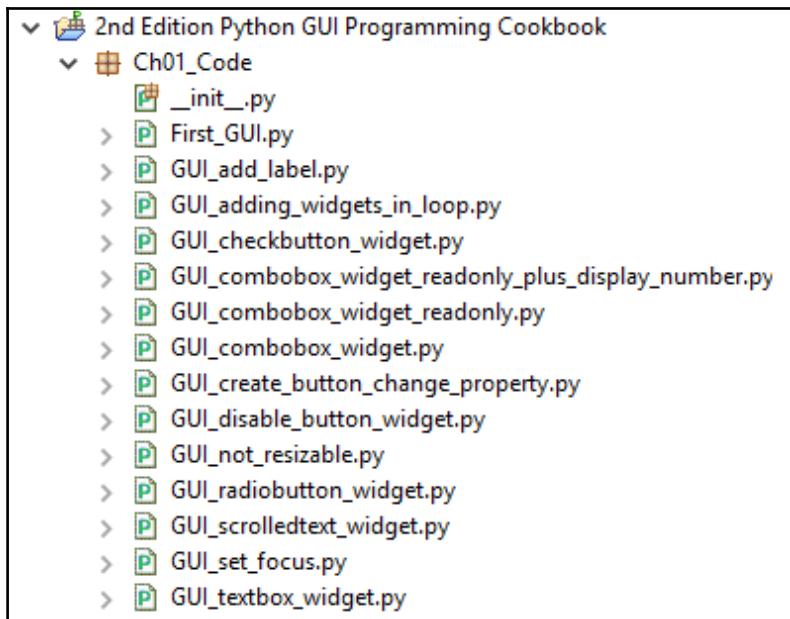
In questo capitolo svilupperemo la nostra prima GUI in Python. Inizieremo con il codice minimo richiesto per creare un'applicazione GUI in esecuzione. Ogni ricetta aggiunge quindi diversi widget al modulo GUI.

Nelle prime due ricette, mostreremo l'intero codice, composto solo da poche righe di codice. Nelle ricette successive mostreremo solo il codice da aggiungere alle ricette precedenti.

Alla fine di questo capitolo, avremo creato un'applicazione GUI funzionante che consiste di etichette, pulsanti, caselle di testo, caselle combinate, pulsanti di controllo in vari stati, nonché pulsanti di opzione che cambiano il colore di sfondo della GUI.

All'inizio di ogni capitolo, mostrerò i moduli Python che appartengono a ciascun capitolo. Farò quindi riferimento ai diversi moduli che appartengono al codice mostrato, studiato ed eseguito.

Ecco la panoramica dei moduli Python (che termina con un'estensione .py) per questo capitolo:



Creazione della nostra prima GUI Python

Python è un linguaggio di programmazione molto potente. Viene fornito con il built-in `tkinter` modulo. In poche righe di codice (quattro, per la precisione) possiamo costruire la nostra prima GUI Python.

Prepararsi

Per seguire questa ricetta, un ambiente di sviluppo Python funzionante è un prerequisito. La GUI IDLE, fornita con Python, è sufficiente per iniziare. IDLE è stato creato utilizzando `tkinter`!

- Tutte le ricette in questo libro sono state sviluppate utilizzando Python 3.6 su un sistema operativo Windows 10 a 64 bit. Non sono stati testati su nessun'altra configurazione. Poiché Python è un linguaggio multiplataforma, il codice di ogni ricetta dovrebbe essere eseguito ovunque.
- Se stai usando un Mac, viene fornito con Python integrato, ma potrebbero mancare alcuni moduli come `tkinter`, che useremo in questo libro.
- Stiamo usando Python 3.6 e il creatore di Python ha scelto intenzionalmente di non renderlo retrocompatibile con Python 2. Se stai usando un Mac o Python 2, potresti dover installare Python 3.6 da www.python.org per eseguire con successo le ricette di questo libro.
- Se desideri davvero eseguire il codice in questo libro su Python 2.7, dovrà apportare alcune modifiche. Ad esempio, `tkinter` in Python 2.x ha una maiuscola. L'istruzione `print` di Python 2.7 è una funzione in Python 3.6 e richiede parentesi.
- Sebbene l'EOL (End Of Life) per il ramo Python 2.x sia stato esteso all'anno 2020, consiglio vivamente di iniziare a utilizzare Python 3.6 e versioni successive.
- Perché aggrapparsi al passato, a meno che non sia proprio necessario? Ecco un collegamento alla Python Enhancement Proposta (PEP) 373 che fa riferimento all'EOL di Python 2: <https://www.python.org/dev/peps/pep-0373/>

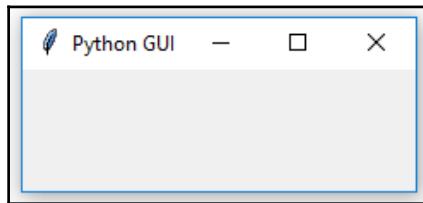


Come farlo...

Ecco le quattro righe di Prima_GUI.py richiesto per creare la GUI risultante:

```
6#=====
7# imports
8#=====
9import tkinter as tk
10
11# Create instance
12win = tk.Tk()
13
14# Add a title
15win.title("Python GUI")
16
17#=====
18# Start GUI
19#=====
20win.mainloop()
```

Esegui questo codice e ammira il risultato:



Come funziona...

Nella riga nove, importiamo il built-in tkinter modulo e alias come tk per semplificare il nostro codice Python. Nella riga 12, creiamo un'istanza di Tk classe chiamando il suo costruttore (le parentesi aggiunte a Tk trasforma la classe in un'istanza). Stiamo usando l'alias tk, quindi non dobbiamo usare la parola più lunga tkinter. Stiamo assegnando l'istanza della classe a una variabile denominata vincere (abbreviazione di finestra). Poiché Python è un linguaggio tipizzato dinamicamente, non abbiamo dovuto dichiarare questa variabile prima di assegnarla e non abbiamo dovuto dargli un tipo specifico. Python deduce il tipo dall'assegnazione di questa istruzione. Python è un linguaggio fortemente tipizzato, quindi ogni variabile ha sempre un tipo. Non è necessario specificarne il tipo in anticipo come in altre lingue. Questo rende Python un linguaggio molto potente e produttivo in cui programmare.

Una piccola nota su classi e tipi:

- In Python, ogni variabile ha sempre un tipo. Non possiamo creare una variabile che non ha un tipo. Tuttavia, in Python, non dobbiamo dichiarare il tipo in anticipo, come dobbiamo fare nel linguaggio di programmazione C.
- Python è abbastanza intelligente da dedurre il tipo. C#, al momento della stesura di questo libro, ha anche questa capacità.
Usando Python, possiamo creare le nostre classi usando il classe parola chiave invece di def parola chiave.
- Per assegnare la classe a una variabile, dobbiamo prima creare un'istanza della nostra classe. Creiamo l'istanza e assegniamo questa istanza alla nostra variabile, ad esempio:
classe AClass(oggetto):

```
    print('Ciao da AClass')
    class_instance = AClass()
```

Ora, la variabile `istanza_classe`, è del Una classe genere. Se questo suona confuso, non preoccuparti. Tratteremo l'OOP nei prossimi capitoli.

Nella riga 15, usiamo la variabile di istanza (`vincere`) della classe per dare un titolo alla nostra finestra tramite il titolo proprietà. Nella riga 20, avviamo il ciclo degli eventi della finestra chiamando il comando `ciclo principale` metodo sull'istanza della classe, `vincere`. Fino a questo punto nel nostro codice, abbiamo creato un'istanza e impostato una proprietà, ma la GUI non verrà visualizzata finché non avvieremo il ciclo di eventi principale.

- Un ciclo di eventi è un meccanismo che fa funzionare la nostra GUI. Possiamo pensarlo come un ciclo infinito in cui la nostra GUI è in attesa che gli vengano inviati eventi. Un clic sul pulsante crea un evento all'interno della nostra GUI, oppure la nostra GUI che viene ridimensionata crea anche un evento.
- Possiamo scrivere tutto il nostro codice GUI in anticipo e nulla verrà visualizzato sullo schermo dell'utente finché non chiamiamo questo ciclo infinito (`win.mainloop()` nel codice precedente).
Il ciclo degli eventi termina quando l'utente fa clic sul rosso X pulsante o un widget che abbiamo programmato per terminare la nostra GUI. Quando il ciclo degli eventi termina, termina anche la nostra GUI.

C'è più...

Questa ricetta ha utilizzato una quantità minima di codice Python per creare il nostro primo programma GUI. Tuttavia, in questo libro useremo l'OOP quando ha senso.

Prevenire il ridimensionamento della GUI

Per impostazione predefinita, una GUI creata utilizzando tkinter può essere ridimensionata. Questo non è sempre l'ideale. I widget che inseriamo nei nostri moduli GUI potrebbero finire per essere ridimensionati in modo improprio, quindi in questa ricetta impareremo come impedire che la nostra GUI venga ridimensionata dall'utente della nostra applicazione GUI.

Prepararsi

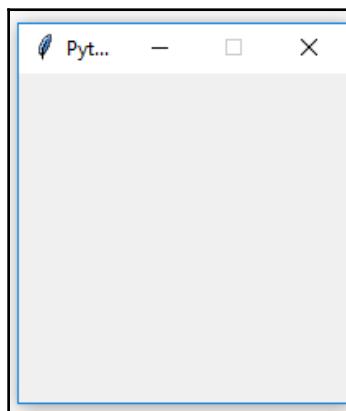
Questa ricetta amplia la precedente, *Creando la nostra prima GUI Python*, quindi un requisito è aver digitato tu stesso la prima ricetta in un tuo progetto, o scaricare il codice da <https://github.com/PacktPublishing/Python-GUI-Programming-Cookbook-Second-Edition/>.

Come farlo...

Stiamo impedendo il ridimensionamento della GUI, guarda:
`GUI_not_resizable.py`

```
6#=====
7# imports
8#####
9import tkinter as tk
10
11# Create instance
12win = tk.Tk()
13
14# Add a title
15win.title("Python GUI")
16
17# Disable resizing the GUI by passing in False/False
18win.resizable(False, False)
19
20# Enable resizing x-dimension, disable y-dimension
21# win.resizable(True, False)
22
23#####
24# Start GUI
25#####
26win.mainloop()
```

L'esecuzione del codice crea questa GUI:



Come funziona...

La riga 18 impedisce il ridimensionamento della GUI di Python.

L'esecuzione di questo codice risulterà in una GUI simile a quella che abbiamo creato nella prima ricetta.

Tuttavia, l'utente non può più ridimensionarlo. Inoltre, nota come il pulsante di ingrandimento nella barra degli strumenti della finestra è disattivato.

Perché questo è importante? Perché una volta aggiunti i widget al nostro modulo, il ridimensionamento può far sembrare la nostra GUI non buona come vorremmo che fosse. Aggiungeremo widget alla nostra GUI nelle prossime ricette.

Il `ridimensionabile()` il metodo è del `Tk()` classe, e passando in (falso, falso), impediamo il ridimensionamento della GUI. Possiamo disabilitare entrambi *X*e *y* dimensioni della GUI da ridimensionare, oppure possiamo abilitare una o entrambe le dimensioni passando dentro Vero o qualsiasi numero diverso da zero. (Vero falso) consentirebbe di *X*-dimensione ma impedisce il ridimensionamento della dimensione *y*.

Abbiamo anche aggiunto commenti al nostro codice in preparazione delle ricette contenute in questo libro.



Negli IDE di programmazione visuale come Visual Studio .NET, i programmatore C# spesso non pensano di impedire all'utente di ridimensionare la GUI che hanno sviluppato in questo linguaggio. Questo crea GUI inferiori. L'aggiunta di questa riga di codice Python può far apprezzare ai nostri utenti la nostra GUI.

Aggiunta di un'etichetta al modulo GUI

Un'etichetta è un widget molto semplice che aggiunge valore alla nostra GUI. Spiega lo scopo degli altri widget, fornendo informazioni aggiuntive. Questo può guidare l'utente al significato di un widget Entry e può anche spiegare i dati visualizzati dai widget senza che l'utente debba inserirvi dati.

Prepararsi

Stiamo ampliando la prima ricetta, *Creazione della nostra prima GUI Python*. Lasceremo la GUI ridimensionabile, quindi non usare il codice della seconda ricetta (o commentare il `win.ridimensionabile` linea fuori).

Come farlo...

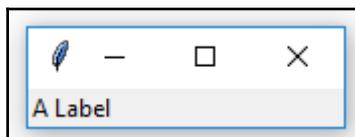
Per aggiungere un Etichetta widget nella nostra GUI, importeremo il ttk modulo da tkinter.

Si prega di notare le due dichiarazioni di importazione. Aggiungi il seguente codice appena sopra win.mainloop(), che si trova in fondo alla prima e alla seconda ricetta:

GUI_add_label.py

```
6#=====
7# imports
8#=====
9import tkinter as tk
10from tkinter import ttk
11
12# Create instance
13win = tk.Tk()
14
15# Add a title
16win.title("Python GUI")
17
18# Adding a Label
19ttk.Label(win, text="A Label").grid(column=0, row=0)
20
21#=====
22# Start GUI
23#=====
24win.mainloop()
```

L'esecuzione del codice aggiunge un'etichetta alla nostra GUI:



Come funziona...

Nella riga 10 del codice precedente, importiamo un modulo separato dal tkinter pacchetto. Il ttk ha alcuni widget avanzati che rendono la nostra interfaccia grafica fantastica. In un senso, ttk è un'estensione all'interno del tkinter pacchetto.

Dobbiamo ancora importare il tkinter pacchetto stesso, ma dobbiamo specificare che ora vogliamo anche usare ttk dal tkinter pacchetto.

ttk sta per *tema tk*. Migliora l'aspetto della nostra GUI.



La riga 19 aggiunge l'etichetta alla GUI, appena prima di chiamare ciclo principale.

Passiamo la nostra istanza della finestra nel ttk.Label costruttore e impostare la proprietà text. Questo diventa il testo nostroEtichetta verrà visualizzato.

Utilizziamo anche il gestore del layout della griglia, che esploreremo in modo molto più approfondito in Capitolo 2, *Gestione del layout*.

Nota come la nostra GUI è diventata improvvisamente molto più piccola rispetto alle ricette precedenti.

Il motivo per cui è diventato così piccolo è che abbiamo aggiunto un widget al nostro modulo. Senza un widget, iltkinter pacchetto utilizza una dimensione predefinita. L'aggiunta di un widget provoca l'ottimizzazione, che generalmente significa utilizzare il minor spazio necessario per visualizzare i widget.

Se rendiamo più lungo il testo dell'etichetta, la GUI si espanderà automaticamente. Tratteremo questa regolazione automatica delle dimensioni del modulo in una ricetta successiva inCapitolo 2, *Gestione del layout*.

C'è più...

Prova a ridimensionare e massimizzare questa GUI con un'etichetta e guarda cosa succede.

Creazione di pulsanti e modifica delle loro proprietà di testo

In questa ricetta, aggiungeremo un widget pulsante e useremo questo pulsante per modificare una proprietà di un altro widget che fa parte della nostra GUI. Questo ci introduce alle funzioni di callback e alla gestione degli eventi in un ambiente GUI Python.

Prepararsi

Questa ricetta amplia la precedente, *Aggiunta di un'etichetta al modulo della GUI*. Puoi scaricare il intero codice da <https://github.com/PacktPublishing/Python-GUI-Programming-Cook book-Second-Edition/>.

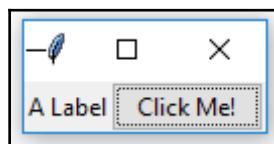
Come farlo...

Aggiungiamo un pulsante che, se cliccato, esegue un'azione. In questa ricetta, aggiorneremo l'etichetta che abbiamo aggiunto nella ricetta precedente e la proprietà del testo del pulsante:

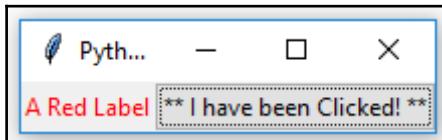
GUI_create_button_change_property.py

```
18 # Adding a Label that will get modified
19 a_label = ttk.Label(win, text="A Label")
20 a_label.grid(column=0, row=0)
21
22 # Button Click Event Function
23@ def click_me():
24     action.configure(text="** I have been Clicked! **")
25     a_label.configure(foreground='red')
26     a_label.configure(text='A Red Label')
27
28 # Adding a Button
29 action = ttk.Button(win, text="Click Me!", command=click_me)
30 action.grid(column=1, row=0)
31
32@ =====
33 # Start GUI
34 =====
35 win.mainloop()
```

La seguente schermata mostra come appare la nostra GUI prima di fare clic sul pulsante:



Dopo aver fatto clic sul pulsante, il colore dell'etichetta è cambiato e anche il testo del pulsante, che può essere visto come segue:



Come funziona...

Nella riga 19, assegniamo l'etichetta a una variabile e nella riga 20 utilizziamo questa variabile per posizionare l'etichetta all'interno del form. Abbiamo bisogno di questa variabile per cambiare le sue proprietà nel cliccammi() funzione. Per impostazione predefinita, questa è una variabile a livello di modulo, quindi possiamo accedervi all'interno della funzione, purché dichiariamo la variabile sopra la funzione che la chiama.

La riga 23 è il gestore dell'evento che viene richiamato una volta che si fa clic sul pulsante. Nella riga 29, creiamo il pulsante e associamo il comando aliccami() funzione.



Le GUI sono guidate dagli eventi. Facendo clic sul pulsante si crea un evento. Leghiamo ciò che accade quando si verifica questo evento nella funzione di callback utilizzando la proprietà command delttk.Button aggeggio. Nota come non usiamo le parentesi, solo il nomecliccammi.

Cambiamo anche il testo dell'etichetta da includere rosso come, nel libro stampato, questo potrebbe non essere altrimenti ovvio. Quando esegui il codice, puoi vedere che il colore cambia davvero.

Le righe 20 e 30 utilizzano entrambe il gestore del layout della griglia, che verrà discusso nel capitolo seguente. Questo allinea sia l'etichetta che il pulsante.

C'è più...

Continueremo ad aggiungere sempre più widget alla nostra GUI e faremo uso di molte proprietà integrate nelle altre ricette del libro.

Widget casella di testo

In tkinter, il tipico widget della casella di testo a una riga si chiama Entry. In questa ricetta, aggiungeremo tale widget Entry alla nostra GUI. Renderemo la nostra etichetta più utile descrivendo cosa sta facendo il widget Entry per l'utente.

Prepararsi

Questa ricetta si basa sul *Creazione di pulsanti e modifica delle loro proprietà di testo* ricetta.

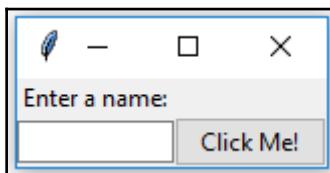
Come farlo...

Dai un'occhiata al seguente codice:

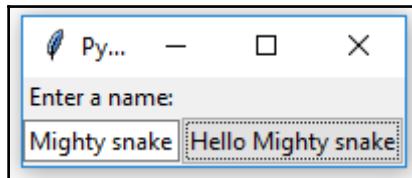
GUI_textbox_widget.py

```
22 # Modified Button Click Function
23 def click_me():
24     action.configure(text='Hello ' + name.get())
25
26 # Changing our Label
27 ttk.Label(win, text="Enter a name:").grid(column=0, row=0)
28
29 # Adding a Text box Entry widget
30 name = tk.StringVar()
31 name_entered = ttk.Entry(win, width=12, textvariable=name)
32 name_entered.grid(column=0, row=1)
```

Ora, la nostra GUI si presenta così:



Dopo aver inserito del testo e aver fatto clic sul pulsante, c'è la seguente modifica nella GUI:



Come funziona...

Nella riga 24, otteniamo il valore del widget Entry. Non abbiamo ancora utilizzato l'OOP, quindi come mai possiamo accedere al valore di una variabile che non è stata ancora dichiarata?

Senza usare le classi OOP, nella codifica procedurale Python, dobbiamo posizionare fisicamente un nome sopra un'istruzione che cerca di usare quel nome. Allora come mai funziona (lo fa)?

La risposta è che l'evento click del pulsante è una funzione di callback e quando un utente fa clic sul pulsante, le variabili a cui si fa riferimento in questa funzione sono note ed esistono.

La vita è bella.

La riga 27 dà alla nostra etichetta un nome più significativo; per ora, descrive la casella di testo sottostante. Abbiamo spostato il pulsante in basso accanto all'etichetta per associare visivamente i due. Stiamo ancora utilizzando il gestore del layout della griglia, che verrà spiegato più dettagliatamente in Capitolo 2, *Gestione del layout*.

La riga 30 crea una variabile, nome. Questa variabile è legata al widget Entry e, nel nostro cliccammi() funzione, siamo in grado di recuperare il valore del widget Entry chiamando otteneri() su questa variabile. Funziona come un incantesimo.

Ora vediamo che mentre il pulsante visualizza l'intero testo che abbiamo inserito (e altro), il widget Entry della casella di testo non si è espanso. Il motivo è che l'abbiamo codificato a una larghezza di 12 nella riga 31.



- Python è un linguaggio tipizzato dinamicamente e deduce il tipo dall'assegnazione. Ciò significa che se assegniamo una stringa a nome variabile, sarà del corda tipo, e se assegniamo un intero a nome, il suo tipo sarà intero.
- Usando tkinter, dobbiamo dichiarare il nome variabile come il tipo tk.StringVar() prima di poterlo utilizzare con successo. Il motivo è che tkinter non è Python. Possiamo usarlo da Python, ma non è lo stesso linguaggio.

Impostare lo stato attivo su un widget e disabilitare i widget

Anche se la nostra GUI sta migliorando, sarebbe più comodo e utile che il cursore appaia nel widget Entry non appena appare la GUI. Qui impariamo come farlo.

Prepararsi

Questa ricetta amplia la ricetta precedente, *Widget casella di testo*.

Come farlo...

Python è davvero fantastico. Tutto quello che dobbiamo fare per impostare il focus su un controllo specifico quando appare la GUI è chiamare il messa a fuoco() metodo su un'istanza di un tkinter widget che abbiamo creato in precedenza. Nel nostro esempio di GUI corrente, abbiamo assegnato il tk.Entry istanza di classe a una variabile denominata, nome_inserito. Ora possiamo concentrarci.

Posiziona il seguente codice appena sopra il codice che si trova nella parte inferiore del modulo e che avvia il ciclo di eventi di Windows principale, come abbiamo fatto nelle ricette precedenti:

GUI_set_focus.py

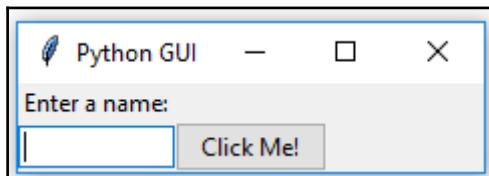
```
29 # Adding a Textbox Entry widget
30 name = tk.StringVar()
31 name_entered = ttk.Entry(win, width=12, textvariable=name)
32 name_entered.grid(column=0, row=1)
33
34 # Adding a Button
35 action = ttk.Button(win, text="Click Me!", command=click_me)
36 action.grid(column=1, row=1)
37
38 name_entered.focus()      # Place cursor into name Entry
39 #=====
40 # Start GUI
41 #=====
42 win.mainloop()
```

Se ricevi degli errori, assicurati di effettuare chiamate alle variabili al di sotto del codice in cui sono dichiarate. Non stiamo ancora utilizzando l'OOP, quindi è ancora necessario. Successivamente, non sarà più necessario farlo.



Su un Mac, potrebbe essere necessario impostare lo stato attivo sulla finestra della GUI prima di poter impostare lo stato attivo sul widget Entry in questa finestra.

L'aggiunta di questa riga (38) di codice Python posiziona il cursore nel nostro widget di immissione di testo, dando il focus al widget di immissione di testo. Non appena appare la GUI, possiamo digitare in questa casella di testo senza dover prima fare clic su di essa.



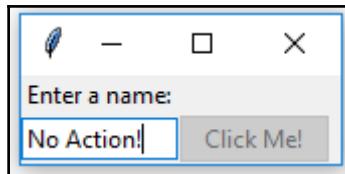


Nota come il cursore ora per impostazione predefinita risiede all'interno della casella di immissione di testo.

Possiamo anche disabilitare i widget. Per fare ciò, imposteremo una proprietà sul widget. Possiamo disabilitare il pulsante aggiungendo questa riga (37 sotto) di codice Python per creare il pulsante:

```
34 # Adding a Button
35 action = ttk.Button(win, text="Click Me!", command=click_me)
36 action.grid(column=1, row=1)
37 action.configure(state='disabled')      # Disable the Button Widget
38
39 name_entered.focus()      # Place cursor into name Entry
```

Dopo aver aggiunto la riga precedente di codice Python, fare clic sul pulsante non crea più alcuna azione:



Come funziona...

Questo codice è autoesplicativo. Mettiamo il focus su un controllo e disabilitiamo un altro widget. Una buona denominazione nei linguaggi di programmazione aiuta ad eliminare lunghe spiegazioni. Più avanti in questo libro, ci saranno alcuni suggerimenti avanzati su come farlo mentre si programma al lavoro o si esercitano le proprie capacità di programmazione a casa.

C'è più...

Sì. Questo è solo il primo capitolo. C'è molto altro in arrivo.

Widget della casella combinata

In questa ricetta, miglioreremo la nostra GUI aggiungendo caselle combinate a discesa che possono avere valori predefiniti iniziali. Sebbene possiamo limitare l'utente solo a determinate scelte, possiamo anche consentire all'utente di digitare ciò che desidera.

Prepararsi

Questa ricetta amplia la ricetta precedente, *Impostare lo stato attivo su un widget e disabilitare i widget*.

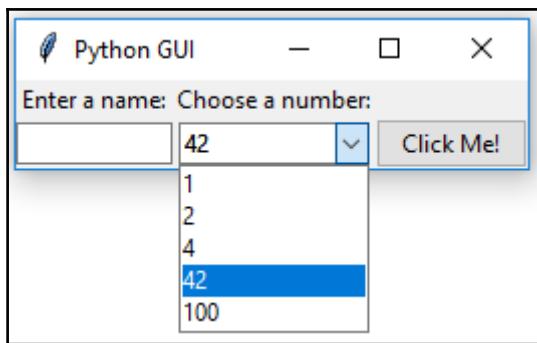
Come farlo...

Inseriamo un'altra colonna tra il widget Entry e il Pulsante widget utilizzando il gestore del layout della griglia. Ecco il codice Python:

GUI_combobox_widget.py

```
31 # Adding a Textbox Entry widget
32 name = tk.StringVar()
33 name_entered = ttk.Entry(win, width=12, textvariable=name)
34 name_entered.grid(column=0, row=1)                                # column 0
35
36 # Adding a Button
37 action = ttk.Button(win, text="Click Me!", command=click_me)
38 action.grid(column=2, row=1)                                         # <= change column to 2
39
40 ttk.Label(win, text="Choose a number:").grid(column=1, row=0)
41 number = tk.StringVar()
42 number_chosen = ttk.Combobox(win, width=12, textvariable=number)
43 number_chosen['values'] = (1, 2, 4, 42, 100)
44 number_chosen.grid(column=1, row=1)                                # <= Combobox in column 1
45 number_chosen.current(0)
46
47 name_entered.focus()      # Place cursor into name Entry
48#=====
49 # Start GUI
50#=====
51 win.mainloop()
```

Questo codice, aggiunto alle ricette precedenti, crea la seguente GUI. Nota come, nella riga 43 del codice precedente, abbiamo assegnato una tupla con valori predefiniti alla casella combinata. Questi valori vengono quindi visualizzati nella casella a discesa. Possiamo anche cambiarli se lo desideriamo (digitando valori diversi quando l'applicazione è in esecuzione):



Come funziona...

La riga 40 aggiunge una seconda etichetta per abbinare la casella combinata appena creata (creata nella riga 42). La riga 41 assegna il valore della casella a una variabile di uno speciale tipo `tkinter.StringVar`, come abbiamo fatto in una ricetta precedente.

La riga 44 allinea i due nuovi controlli (etichetta e casella combinata) all'interno del nostro precedente layout della GUI e la riga 45 assegna un valore predefinito da visualizzare quando la GUI diventa visibile per la prima volta. Questo è il primo valore `dinumero_scelto['valori']` tupla, la stringa "1". Non abbiamo inserito le virgolette attorno alla nostra tupla di interi nella riga 43, ma sono state convertite in stringhe perché, nella riga 41, abbiamo dichiarato che i valori sono della `tk.StringVar` genere.

La schermata precedente mostra la selezione effettuata dall'utente come 42. Questo valore viene assegnato a numero variabile.

C'è più...

Se vogliamo limitare l'utente a poter selezionare solo i valori che abbiamo programmato nel Casella combinata, possiamo farlo passando il stato *proprietà* nel costruttore. Modificare la riga 42 come segue:

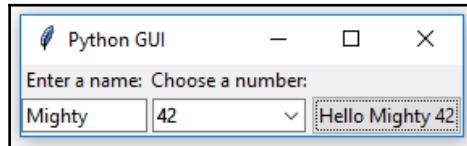
GUI_combobox_widget_READONLY_plus_display_number.py

```
40 ttk.Label(win, text="Choose a number:").grid(column=1, row=0)
41 number = tk.StringVar()
42 number_chosen = ttk.Combobox(win, width=12, textvariable=number, state='readonly')
43 number_chosen['values'] = (1, 2, 4, 42, 100)
44 number_chosen.grid(column=1, row=1)
45 number_chosen.current(0)
```

Ora, gli utenti non possono più digitare valori nel Casella combinata. Possiamo visualizzare il valore scelto dall'utente aggiungendo la seguente riga di codice al nostro *Pulsante Clic Evento Richiamata* funzione:

```
22 # Modified Button Click Function
23 def click_me():
24     action.configure(text='Hello ' + name.get() + ' ' +
25     number_chosen.get())
```

Dopo aver scelto un numero, inserito un nome e quindi fatto clic sul pulsante, otteniamo il seguente risultato della GUI, che ora mostra anche il numero selezionato:



Creazione di un pulsante di spunta con diversi stati iniziali

In questa ricetta, aggiungeremo tre widget di pulsanti di controllo, ciascuno con uno stato iniziale diverso.

Prepararsi

Questa ricetta amplia la ricetta precedente, *Widget della casella combinata*.

Come farlo...

Stiamo creando tre widget di pulsanti di controllo che differiscono nei loro stati. Il primo è disabilitato e ha un segno di spunta. L'utente non può rimuovere questo segno di spunta poiché il widget è disabilitato.

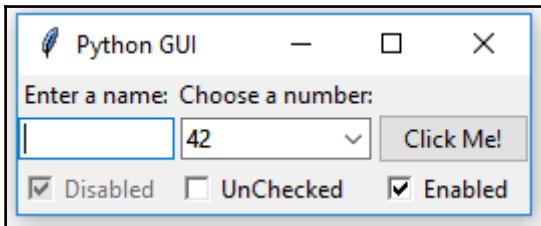
Il secondo pulsante di spunta è abilitato e, per impostazione predefinita, non contiene alcun segno di spunta, ma l'utente può fare clic su di esso per aggiungere un segno di spunta.

Il terzo pulsante di controllo è sia abilitato che selezionato per impostazione predefinita. Gli utenti possono deselectare e ricontrizzare il widget tutte le volte che lo desiderano. Guarda il seguente codice:

GUI_checkbutton_widget.py

```
35 # Adding a Button
36 action = ttk.Button(win, text="Click Me!", command=click_me)
37 action.grid(column=2, row=1)
38
39 # Creating three checkbuttons
40 ttk.Label(win, text="Choose a number:").grid(column=1, row=0)
41 number = tk.StringVar()
42 number_chosen = ttk.Combobox(win, width=12, textvariable=number, state='readonly')
43 number_chosen['values'] = (1, 2, 4, 42, 100)
44 number_chosen.grid(column=1, row=1)
45 number_chosen.current(0)
46
47 chVarDis = tk.IntVar()
48 check1 = tk.Checkbutton(win, text="Disabled", variable=chVarDis, state='disabled')
49 check1.select()
50 check1.grid(column=0, row=4, sticky=tk.W)
51
52 chVarUn = tk.IntVar()
53 check2 = tk.Checkbutton(win, text="UnChecked", variable=chVarUn)
54 check2.deselect()
55 check2.grid(column=1, row=4, sticky=tk.W)
56
57 chVarEn = tk.IntVar()
58 check3 = tk.Checkbutton(win, text="Enabled", variable=chVarEn)
59 check3.select()
60 check3.grid(column=2, row=4, sticky=tk.W)
61
62 name_entered.focus()      # Place cursor into name Entry
63 #=====
64 # Start GUI
65 #=====
66 win.mainloop()
```

L'esecuzione del nuovo codice comporta la seguente GUI:



Come funziona...

Nelle righe 47, 52 e 57 creiamo tre variabili di IntVar genere. Nella riga che segue ciascuna di queste variabili, creiamo aPulsante di spunta, passando in queste variabili. Manterranno lo stato delPulsante di spunta (non selezionato o selezionato). Per impostazione predefinita, è 0 (deselezionato) o 1 (selezionato), quindi il tipo della variabile è atkinter numero intero.

Mettiamo questi Pulsante di controllo widgets nella nostra finestra principale, quindi il primo argomento passato al costruttore è il genitore del widget, nel nostro caso, vincere. diamo a ciascuno Pulsante di controllo widget un'etichetta diversa tramite la sua testo proprietà.

Impostare la proprietà sticky della griglia su tk.W significa che il widget sarà allineato a ovest della griglia. Questo è molto simile alla sintassi Java e significa che sarà allineato a sinistra. Quando ridimensioniamo la nostra GUI, il widget rimarrà sul lato sinistro e non verrà spostato verso il centro della GUI.

Le righe 49 e 59 mettono un segno di spunta in Pulsante di controllo widget chiamando il Selezionare() metodo su questi due Pulsante di controllo istanze di classe.

Continuiamo a organizzare i nostri widget utilizzando il gestore del layout della griglia, che verrà spiegato più dettagliatamente in Capitolo 2, *Gestione del layout*.

Utilizzo dei widget dei pulsanti di opzione

In questa ricetta creeremo tre tkinter Pulsante radio widget. Aggiungeremo anche del codice che cambia il colore del modulo principale, a seconda di qualePulsante radio è selezionato.

Prepararsi

Questa ricetta amplia la ricetta precedente, *Creazione di un pulsante di spunta con diversi stati iniziali*.

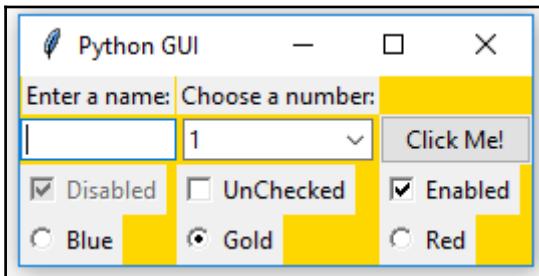
Come farlo...

Aggiungiamo il seguente codice alla ricetta precedente:

GUI_radiobutton_widget.py

```
74 # Radiobutton Globals
75 COLOR1 = "Blue"
76 COLOR2 = "Gold"
77 COLOR3 = "Red"
78
79 # Radiobutton Callback
80 def radCall():
81     radSel=radVar.get()
82     if radSel == 1: win.configure(background=COLOR1)
83     elif radSel == 2: win.configure(background=COLOR2)
84     elif radSel == 3: win.configure(background=COLOR3)
85
86 # create three Radiobuttons using one variable
87 radVar = tk.IntVar()
88
89 rad1 = tk.Radiobutton(win, text=COLOR1, variable=radVar, value=1, command=radCall)
90 rad1.grid(column=0, row=5, sticky=tk.W, columnspan=3)
91
92 rad2 = tk.Radiobutton(win, text=COLOR2, variable=radVar, value=2, command=radCall)
93 rad2.grid(column=1, row=5, sticky=tk.W, columnspan=3)
94
95 rad3 = tk.Radiobutton(win, text=COLOR3, variable=radVar, value=3, command=radCall)
96 rad3.grid(column=2, row=5, sticky=tk.W, columnspan=3)
97
98 name_entered.focus()      # Place cursor into name Entry
99 #=====
100 # Start GUI
101 #=====
102 win.mainloop()
```

Eseguendo questo codice e selezionando il Pulsante radio di nome **Oro** crea la seguente finestra:



Come funziona...

Nelle righe 75-77, creiamo alcune variabili globali a livello di modulo che utilizzeremo nella creazione di ciascun pulsante di opzione e nella funzione di callback che crea l'azione di cambiare il colore di sfondo del form principale (usando la variabile di istanza vincere).

Utilizziamo variabili globali per semplificare la modifica del codice. Assegnando il nome del colore a una variabile e utilizzando questa variabile in più punti, possiamo facilmente sperimentare colori diversi. Invece di eseguire una ricerca e sostituzione globale di una stringa codificata (che è soggetta a errori), è sufficiente modificare una riga di codice e tutto il resto funzionerà. Questo è noto come**principio ASCIUTTO**, che sta per **Non ripeterti**. Questo è un concetto OOP che useremo nelle ricette successive del libro.



I nomi dei colori che stiamo assegnando alle variabili (COLORE1, COLORE2 ...) siamo tkinter parole chiave (tecnicamente, sono *nomi simbolici*). Se noi usa nomi che non lo sono tkinter parole chiave di colore, il codice non funzionerà.

La riga 80 è la *funzione di richiamata* che cambia lo sfondo del nostro modulo principale (vincere) a seconda della selezione dell'utente.

Nella riga 87 creiamo a tk.IntVar variabile. Ciò che è importante è che creiamo solo una variabile che deve essere utilizzata da tutti e tre i pulsanti di opzione. Come si può vedere dallo screenshot, non importa qualePulsante radio selezioniamo, tutti gli altri verranno automaticamente deselezionati per noi.

Le righe da 89 a 96 creano i tre radio button, assegnandoli al form principale, passandoci la variabile da utilizzare nella funzione di callback che crea l'azione di cambiare lo sfondo della nostra finestra principale.



Anche se questa è la prima ricetta che cambia il colore di un widget, onestamente, sembra un po' brutto. Una gran parte delle seguenti ricette in questo libro spiega come rendere la nostra GUI davvero sorprendente.

C'è più...

Ecco un piccolo esempio dei nomi di colori simbolici disponibili che puoi consultare su ufficiale **tcl** pagina di manuale su <http://www.tcl.tk/man/tcl8.5/TkCmd/colors.htm>.

Nome	Rosso	Verde	blu
alice blu	240	248	255
AliceBlue	240	248	255
Blu	0	0	255
Oro	255	215	0
Rosso	255	0	0

Alcuni nomi creano lo stesso colore, quindi alice blu crea lo stesso colore di AliceBlue. In questa ricetta abbiamo usato i nomi simbolici Blu, Oro, e Rosso.

Utilizzo di widget di testo scorrevole

Testo scorrevole i widget sono molto più grandi del semplice Iscrizione widget e si estendono su più righe. Sono widget come Blocco note e righe a capo, che abilitano automaticamente le barre di scorrimento verticali quando il testo diventa più grande dell'altezza del Testo scorrevole aggeggio.

Prepararsi

Questa ricetta amplia la ricetta precedente, *Utilizzo dei widget dei pulsanti di opzione*. Puoi scaricare il codice per ogni capitolo di questo libro da [https://github.com/PacktPublishing/Python-G UI-Programming-Cookbook-Second-Edition/](https://github.com/PacktPublishing/Python-GUI-Programming-Cookbook-Second-Edition/).

Come farlo...

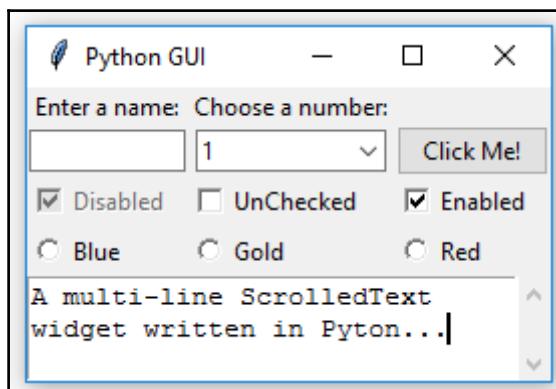
Aggiungendo le seguenti righe di codice, creiamo a Testo scorrevole aggeggio:

GUI_scrolledtext_widget.py

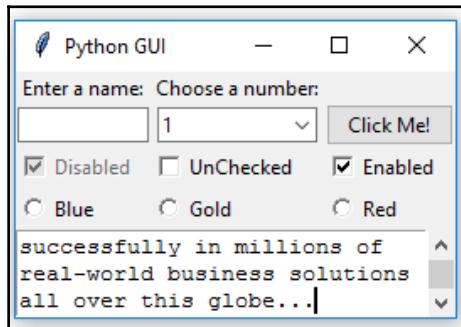
```
6#=====
7# imports
8#=====
9import tkinter as tk
10from tkinter import ttk
11from tkinter import scrolledtext

99# Using a scrolled Text control
100scrol_w = 30
101scrol_h = 3
102scr = scrolledtext.ScrolledText(win, width=scrol_w, height=scrol_h, wrap=tk.WORD)
103scr.grid(column=0, columnspan=3)
104
105name_entered.focus()      # Place cursor into name Entry
106#=====
107# Start GUI
108#=====
109win.mainloop()
```

Possiamo effettivamente digitare nel nostro widget e, se digitiamo abbastanza parole, le righe si avvolgeranno automaticamente:



Una volta digitate più parole dell'altezza che il widget può visualizzare, la barra di scorrimento verticale viene abilitata. Tutto questo funziona immediatamente senza che dobbiamo scrivere altro codice per raggiungere questo obiettivo:



Come funziona...

Nella riga 11, importiamo il modulo che contiene il Testo scorrevole classe widget. Aggiungi questo nella parte superiore del modulo, appena sotto gli altri due importare dichiarazioni.

Le righe 100 e 101 definiscono la larghezza e l'altezza del Testo scorrevole widget che stiamo per creare. Questi sono valori hardcoded che stiamo passando a Testo scorrevole costruttore di widget nella riga 102.

Questi valori sono *numeri magici* trovato dalla sperimentazione per funzionare bene. Potresti sperimentare cambiandoscol_w da 30 a 50 e osserva l'effetto!

Nella riga 102, stiamo anche impostando una proprietà sul widget passando in wrap=tk.WORD.

Impostando il avvolgere proprietà a tk.WORD stiamo dicendo a Testo scorrevole widget per spezzare le righe per parole in modo da non avvolgerci all'interno di una parola. L'opzione predefinita è tk.CHAR, che avvolge qualsiasi carattere indipendentemente dal fatto che ci troviamo nel mezzo di una parola.

La seconda schermata mostra che la barra di scorrimento verticale si è spostata verso il basso perché stiamo leggendo un testo più lungo che non si adatta completamente al x, y dimensioni del SrolledText controllo che abbiamo creato.

Impostazione del ampiezza delle colonne proprietà del widget della griglia per 3 per il SrolledText widget fa in modo che questo widget occupi tutte e tre le colonne. Se non impostiamo questa proprietà, il nostro SrolledText widget risiederebbe solo nella colonna uno, che non è quello che vogliamo.

Aggiunta di più widget in un ciclo

Finora abbiamo creato diversi widget dello stesso tipo (ad esempio, Pulsante radio) sostanzialmente copiando e incollando lo stesso codice e poi modificando le variazioni (ad esempio il numero di colonna). In questa ricetta, iniziamo a rifactorizzare il nostro codice per renderlo meno ridondante.

Prepararsi

Stiamo effettuando il refactoring di alcune parti del codice della ricetta precedente, *Utilizzando widget di testo scorrevole*, quindi hai bisogno di quel codice per applicare questa ricetta.

Come farlo...

Ecco come refactoriamo il nostro codice:

GUI_aggiungendo_widgets_in_loop.py

```
76 # First, we change our Radiobutton global variables into a list
77 colors = ["Blue", "Gold", "Red"]
78
79# We have also changed the callback function to be zero-based, using the list
80 # instead of module-level global variables
81 # Radiobutton Callback
82def radCall():
83     radSel=radVar.get()
84     if radSel == 0: win.configure(background=colors[0]) # now zero-based
85     elif radSel == 1: win.configure(background=colors[1]) # and using list
86     elif radSel == 2: win.configure(background=colors[2])
87
88 # create three Radiobuttons using one variable
89 radVar = tk.IntVar()
90
91 # Next we are selecting a non-existing index value for radVar
92 radVar.set(99)
93
94 # Now we are creating all three Radiobutton widgets within one loop
95 for col in range(3):
96     curRad = tk.Radiobutton(win, text=colors[col], variable=radVar,
97                            value=col, command=radCall)
98     curRad.grid(column=col, row=5, sticky=tk.W)
99
```

L'esecuzione di questo codice creerà la stessa finestra di prima, ma il nostro codice è molto più pulito e più facile da mantenere. Questo ci aiuterà quando espandiamo la nostra GUI nelle prossime ricette.

Come funziona...

Nella riga 77, abbiamo trasformato le nostre variabili globali in un elenco.

Nella riga 89, impostiamo un valore predefinito su tk.IntVar variabile che abbiamo chiamato radVar. Questo è importante perché, mentre nella ricetta precedente avevamo impostato il valore per Pulsante radio widget a partire da 1, nel nostro nuovo ciclo è molto più conveniente usare l'indicizzazione a base zero di Python. Se non abbiamo impostato il valore predefinito su un valore al di fuori dell'intervallo del nostro Pulsante radio widget, uno dei pulsanti di opzione verrebbe selezionato quando viene visualizzata la GUI. Anche se questo di per sé potrebbe non essere così male, *non attiverebbe la richiamata* e ci ritroveremmo con un pulsante di opzione selezionato che non fa il suo lavoro (cioè cambia il colore del modulo di vincita principale).

Nella riga 95, sostituiamo le tre creazioni precedentemente hardcoded del Pulsante radio widget con un ciclo che fa lo stesso. È solo più conciso (meno righe di codice) e molto più gestibile. Ad esempio, se vogliamo creare 100 anziché solo trePulsante radio widget, tutto ciò che dobbiamo cambiare è il numero all'interno dell'operatore range di Python. Non dovremmo digitare o copiare e incollare 97 sezioni di codice duplicato, solo un numero.

La riga 82 mostra la funzione di richiamata modificata.

C'è più...

Questa ricetta conclude il primo capitolo di questo libro. Tutte le seguenti ricette in tutti i prossimi capitoli si baseranno sulla GUI che abbiamo costruito finora, migliorandola notevolmente.

2

Gestione del layout

In questo capitolo, organizzeremo la nostra GUI utilizzando Python 3.6 e versioni successive. Tratteremo le seguenti ricette:

- Disposizione di più etichette all'interno di un widget cornice etichetta
- Utilizzo della spaziatura interna per aggiungere spazio attorno ai widget
- Come i widget espandono dinamicamente la GUI
- Allineare i widget della GUI incorporando i frame all'interno dei frame Creazione delle barre dei menu
- Creazione di widget a schede Utilizzo
- del gestore del layout della griglia

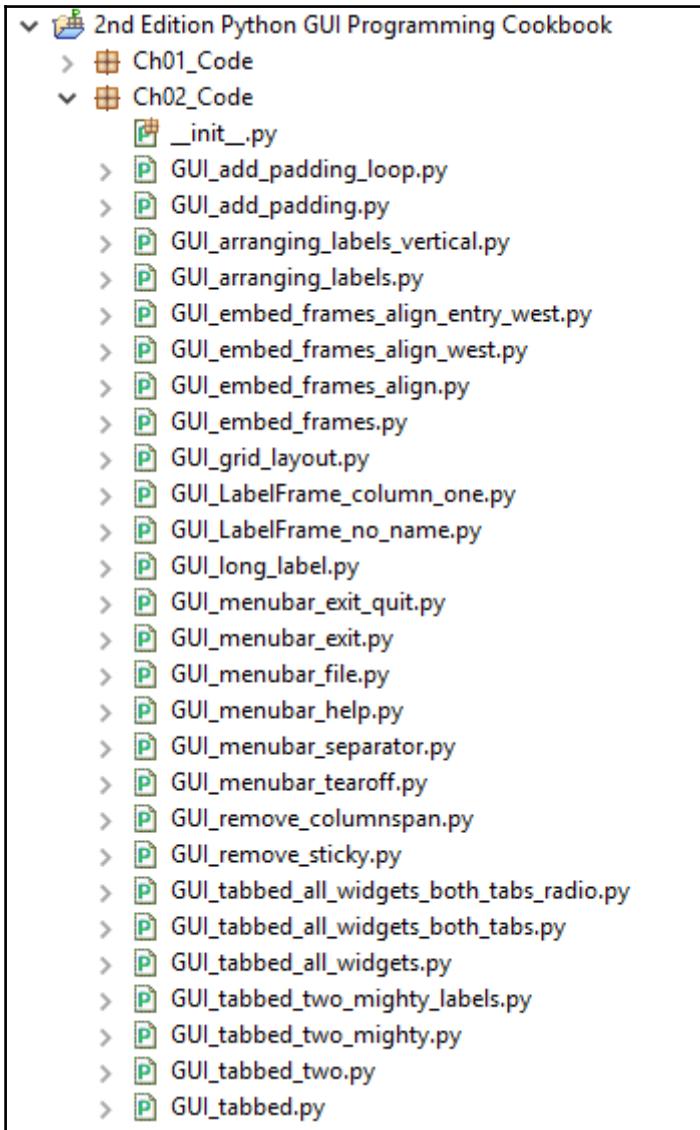
introduzione

In questo capitolo, esploreremo come disporre i widget all'interno dei widget per creare la nostra GUI Python. L'apprendimento dei fondamenti della progettazione del layout della GUI ci consentirà di creare GUI di bell'aspetto. Ci sono alcune tecniche che ci aiuteranno a realizzare questo progetto di layout.

Il gestore del layout della griglia è uno degli strumenti di layout più importanti integrati in tkinter che utilizzeremo.

Possiamo creare facilmente barre dei menu, controlli a schede (noti anche come Notebook) e molti altri widget utilizzando tkinter.

Ecco una panoramica dei moduli Python utilizzati in questo capitolo:



Disposizione di più etichette all'interno di un widget cornice etichetta

Il EtichettaFrame widget ci permette di progettare la nostra GUI in modo organizzato. Stiamo ancora usando il gestore del layout della griglia come nostro principale strumento di progettazione del layout, ma usando EtichettaFrame widget, abbiamo molto più controllo sul design della nostra GUI.

Prepararsi

Inizieremo ad aggiungere sempre più widget alla nostra GUI e renderemo la GUI completamente funzionante nelle prossime ricette. Qui, inizieremo a usare il EtichettaFrame aggiuggo. Riutilizzeremo la GUI dalla ricetta, *Aggiunta di più widget in un ciclo* nel Capitolo 1, *Creazione del modulo GUI e aggiunta di widget*.

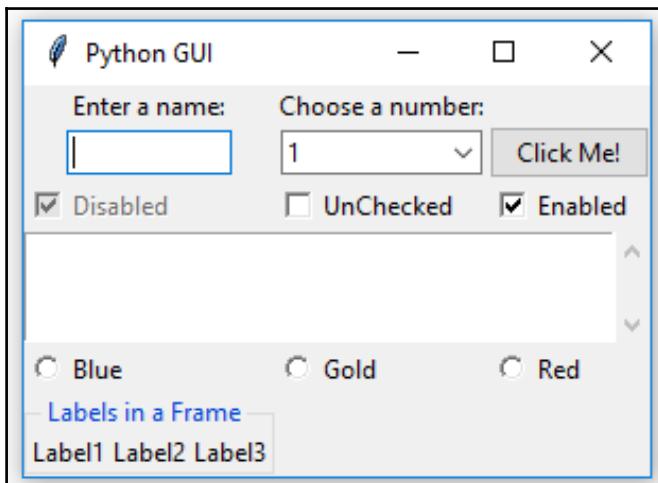
Come farlo...

Aggiungi il seguente codice appena sopra il ciclo dell'evento principale verso la parte inferiore del modulo Python:

GUI_LabelFrame_column_one.py

```
108 # Create a container to hold labels
109 buttons_frame = ttk.LabelFrame(win, text=' Labels in a Frame ')
110 buttons_frame.grid(column=0, row=7)
111 # buttons_frame.grid(column=1, row=7)                      # now in col 1
112
113 # Place labels into the container element
114 ttk.Label(buttons_frame, text="Label1").grid(column=0, row=0, sticky=tk.W)
115 ttk.Label(buttons_frame, text="Label2").grid(column=1, row=0, sticky=tk.W)
116 ttk.Label(buttons_frame, text="Label3").grid(column=2, row=0, sticky=tk.W)
117
118 name_entered.focus()      # Place cursor into name Entry
119 #=====
120 # Start GUI
121 #=====
122 win.mainloop()
```

L'esecuzione del codice comporterà l'aspetto della GUI come segue:



Decommenta la riga 111 e nota il diverso allineamento di EtichettaFrame.



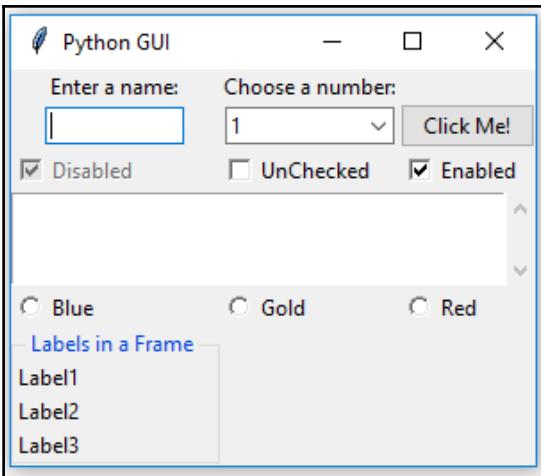
Possiamo facilmente allineare le etichette verticalmente modificando il nostro codice, come mostrato nella schermata successiva:

```
113 # Place labels into the container element
114 ttk.Label(buttons_frame, text="Label1").grid(column=0, row=0)
115 ttk.Label(buttons_frame, text="Label2").grid(column=0, row=1)
116 ttk.Label(buttons_frame, text="Label3").grid(column=0, row=2)
117
118 for child in buttons_frame.winfo_children():
```

Nota che l'unica modifica che dovevamo fare era nella numerazione delle colonne e delle righe.



Ora la GUI EtichettaFrame appare come segue:



Come funziona...

Nella riga 109, creiamo il nostro primo ttk LabelFrame widget e assegna l'istanza risultante al pulsanti_frame variabile. Il contenitore genitore è vincere, la nostra finestra principale.

Nelle righe da 114 a 116, creiamo etichette e le posizioniamo dentro EtichettaFrame. pulsanti_frame è il genitore delle etichette. Utilizziamo l'importante strumento di layout della griglia per disporre le etichette all'interno del EtichettaFrame. Le proprietà di colonna e riga di questo gestore di layout ci danno il potere di controllare il nostro layout della GUI.



Il capostipite delle nostre etichette è il pulsanti_frame variabile di istanza di EtichettaFrame, non il vincere variabile di istanza della finestra principale. Possiamo vedere l'inizio di una gerarchia di layout qui.

Possiamo vedere quanto sia facile cambiare il nostro layout tramite le proprietà di colonna e riga. Nota come cambiamo la colonna in0, e come sovrapponiamo le nostre etichette verticalmente numerando i valori delle righe in sequenza.



Il nome ttk sta per *tema tk*. Il set di widget a tema tk è stato introdotto in Tk 8.5.

C'è più...

In una ricetta più avanti in questo capitolo, inseriremo EtichettaFrame widget all'interno EtichettaFrame widget, annidandoli per controllare il nostro layout della GUI.

Utilizzo della spaziatura interna per aggiungere spazio intorno ai widget

La nostra GUI è stata creata bene. Successivamente, miglioreremo gli aspetti visivi dei nostri widget aggiungendo un po' di spazio intorno a loro, in modo che possano respirare.

Prepararsi

Sebbene tkinter potesse avere la reputazione di creare brutte GUI, questo è cambiato radicalmente dalla versione 8.5. Devi solo sapere come utilizzare gli strumenti e le tecniche disponibili. Questo è quello che faremo dopo.

tkinter versione 8.6 viene fornito con Python 3.6.



Come farlo...

Viene mostrato per primo il modo procedurale di aggiungere spazi intorno ai widget, quindi utilizzeremo un ciclo per ottenere la stessa cosa in un modo molto migliore.

Nostro EtichettaFrame sembra un po' stretto in quanto si fonde con la finestra principale verso il basso. Risolviamolo ora.

Modifica la riga 110 dello snippet di codice da `GUI_LabelFrame_column_one.py` nella ricetta precedente aggiungendo `padx` e `pady`. Trovi il codice anche in:

`GUI_add_padding.py`

```
buttons_frame.grid(column=0, row=7, padx=20, pady=40)
```

Ora, il nostro EtichettaFrame trova un po' di respiro:



Come funziona...

In tkinter, l'aggiunta di spazio orizzontalmente e verticalmente viene eseguita utilizzando le proprietà integrate denominate padx e pady. Questi possono essere usati per aggiungere spazio intorno a molti widget, migliorando rispettivamente gli allineamenti orizzontali e verticali. Abbiamo hardcoded 20 pixel di spazio a sinistra e a destra di EtichettaFrame, e abbiamo aggiunto 40 pixel nella parte superiore e inferiore del fotogramma. Ora il nostro EtichettaFrame si distingue meglio di prima.

Lo screenshot precedente mostra solo la modifica rilevante.

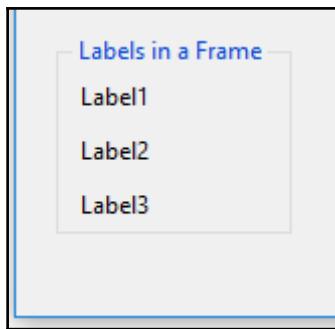


Possiamo usare un ciclo per aggiungere spazio attorno alle etichette contenute all'interno EtichettaTelaio:

GUI_add_padding_loop.py

```
113 # Place labels into the container element
114 ttk.Label(buttons_frame, text="Label1").grid(column=0, row=0)
115 ttk.Label(buttons_frame, text="Label2").grid(column=0, row=1)
116 ttk.Label(buttons_frame, text="Label3").grid(column=0, row=2)
117
118 for child in buttons_frame.winfo_children():
119     child.grid_configure(padx=8, pady=4)
120
121 name_entered.focus()      # Place cursor into name Entry
122 #####
123 # Start GUI
124 #####
125 win.mainloop()
```

Ora, le etichette all'interno del EtichettaFrame anche i widget hanno dello spazio intorno a loro:



Il grid_configure() La funzione ci consente di modificare gli elementi dell'interfaccia utente prima che il ciclo principale li visualizzi. Quindi, invece di codificare i valori quando creiamo per la prima volta un widget, possiamo lavorare sul nostro layout e quindi organizzare la spaziatura verso la fine del nostro file, appena prima che venga creata la GUI. Questa è una tecnica pulita da conoscere.

Il winfo_children() la funzione restituisce un elenco di tutti i figli appartenenti al pulsanti_frame variabile. Questo ci consente di scorrerli e assegnare il riempimento a ciascuna etichetta.

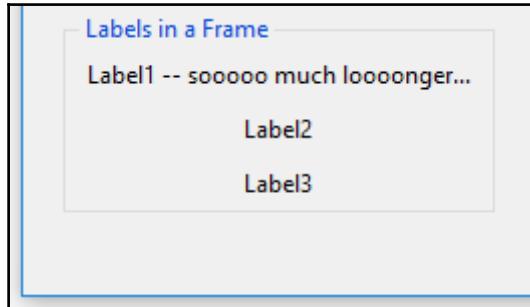


Una cosa da notare è che la spaziatura a destra delle etichette non è realmente visibile. Questo perché il titolo diEtichettaFrame è più lungo dei nomi delle etichette. Possiamo sperimentare questo allungando i nomi delle etichette.

Considera le seguenti modifiche al codice come in GUI_long_label.py:

```
113 # Place labels into the container element - vertically with long label
114 ttk.Label(buttons_frame, text="Label1 -- sooooo much loooonger...").grid(column=0, row=0)
115 ttk.Label(buttons_frame, text="Label2").grid(column=0, row=1)
116 ttk.Label(buttons_frame, text="Label3").grid(column=0, row=2)
```

Ora, la nostra GUI appare come mostrato nello screenshot seguente. Nota come ora c'è dello spazio aggiunto a destra dell'etichetta lunga accanto ai punti. L'ultimo punto non si tocca EtichettaFrame, che altrimenti avrebbe, senza lo spazio aggiunto:

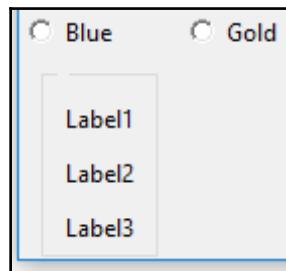


Possiamo anche rimuovere il nome di EtichettaFrame per vedere l'effetto padx ha sulla posizione delle nostre etichette:

GUI_LabelFrame_no_name.py

```
109 buttons_frame = ttk.LabelFrame(win, text="") # no LabelFrame name
```

Impostando il testo proprietà a una stringa vuota, rimuoviamo il nome che era stato precedentemente visualizzato per EtichettaTelaio:



Come i widget espandono dinamicamente la GUI

Potresti aver notato negli screenshot precedenti ed eseguendo il codice che i widget hanno la capacità di estendersi allo spazio di cui hanno bisogno per visualizzare visivamente il loro testo.



- Java ha introdotto il concetto di gestione del layout dinamico della GUI. In confronto, gli IDE di sviluppo visivo, come VS.NET, dispongono la GUI in modo visivo e fondamentalmente codificano l'hardcode *X e si'* coordinate degli elementi dell'interfaccia utente.
- Usando tkinter, questa capacità dinamica crea sia un vantaggio che una piccola sfida perché a volte la nostra GUI si espande dinamicamente quando vorremmo che non fosse così dinamica! Bene, siamo programmati Python dinamici, quindi possiamo capire come sfruttare al meglio questo fantastico comportamento!

Prepararsi

All'inizio della ricetta precedente, *Utilizzo della spaziatura interna per aggiungere spazio intorno ai widget*, abbiamo aggiunto un EtichettaFrame aggeggio. Questo ha spostato alcuni dei nostri controlli al centro della colonna0. Potremmo non volere questa modifica nel nostro layout della GUI. Successivamente, esploreremo alcuni modi per risolverlo.

Come farlo...

Diventiamo prima consapevoli dei sottili dettagli che stanno avvenendo nel nostro layout della GUI per capirlo meglio.

Stiamo utilizzando il widget del gestore del layout della griglia e dispone i nostri widget in una griglia a base zero. È molto simile a un foglio di calcolo Excel o una tabella di database.

Di seguito è riportato un esempio di un gestore di layout di griglia con due righe e tre colonne:

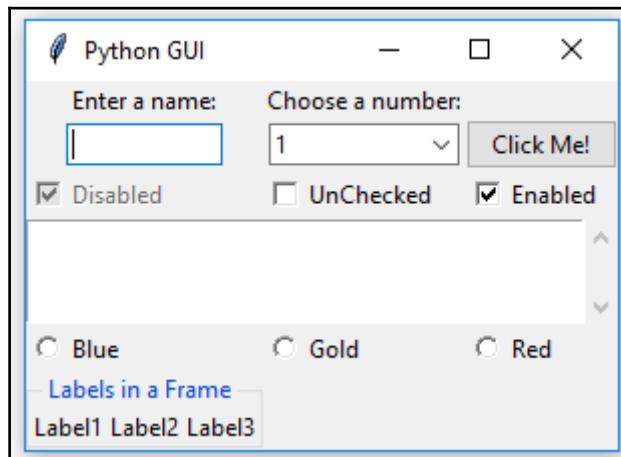
Riga 0; Col 0	Riga 0; Col 1	Riga 0; Col 2
Riga 1; Col 0	Riga 1; Col 1	Riga 1; Col 2

Utilizzando il gestore del layout della griglia, ciò che accade è che la larghezza di una determinata colonna è determinata dal nome o widget più lungo in quella colonna. Questo ha effetto su tutte le righe.

Aggiungendo il nostro EtichettaFrame widget e dandogli un titolo che è più lungo di alcuni widget di dimensioni hardcoded, come l'etichetta in alto a sinistra e la voce di testo sotto di essa, spostiamo dinamicamente quei widget al centro della colonna 0, aggiungendo spazio sul lato sinistro e destro di quei widget.

Per inciso, perché abbiamo usato il appiccicoso proprietà per il Pulsante di controllo e Testo scorrevole widget, questi rimangono attaccati al lato sinistro del frame.

Diamo un'occhiata più in dettaglio allo screenshot della prima ricetta di questo capitolo, *Disposizione di più etichette all'interno di un widget cornice etichetta*:



Abbiamo aggiunto il seguente codice per creare EtichettaFrame e quindi posizionate le etichette in questo frame:

```
108 # Create a container to hold labels
109 buttons_frame = ttk.LabelFrame(win, text='Labels in a Frame ')
110 buttons_frame.grid(column=0, row=7)
```

Poiché la proprietà text di the EtichettaFrame, che viene visualizzato come titolo del EtichettaFrame, è più lungo di entrambi i nostri **Inserisci un nome:** label e la voce della casella di testo sottostante, questi due widget sono centrati dinamicamente all'interno della nuova larghezza della colonna 0.

Il Pulsante di controllo e Pulsante radio widget in colonna 0 non è stato centrato perché abbiamo usato il appiccicoso=tk.W proprietà quando abbiamo creato quei widget.

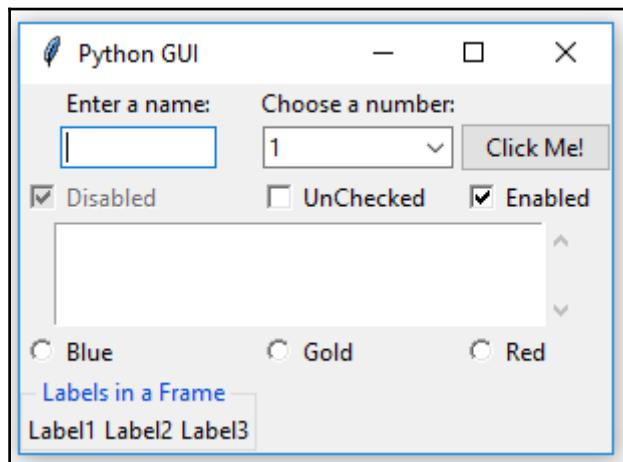
Per il Testo scorrevole widget, abbiamo usato appiccicoso=tk.NOI, che lega il widget sia al lato ovest (alias sinistro) che est (alias destro) del frame.

Rimuoviamo il appiccicoso proprietà dal Testo scorrevole widget e osserva l'effetto di questa modifica:

GUI_remove_sticky.py

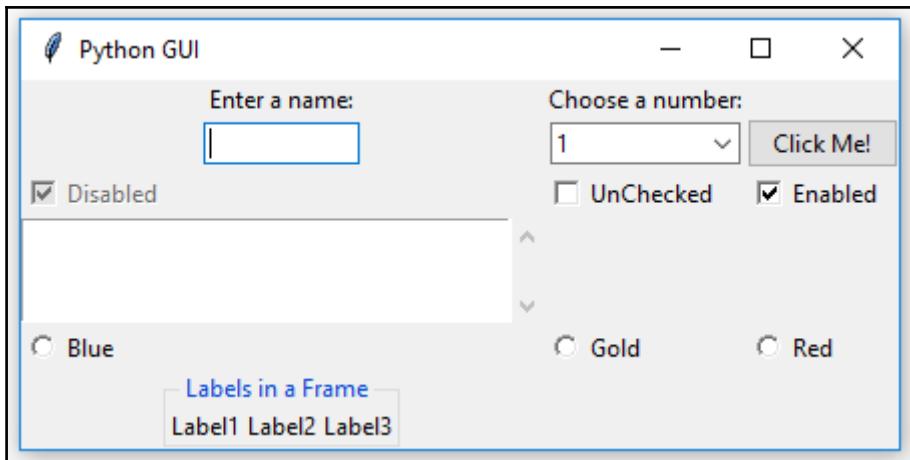
```
# Using a scrolled Text control
scrol_w = 30
scrol_h = 3
scr = scrolledtext.ScrolledText(win, width=scrol_w, height=scrol_h, wrap=tk.WORD)
#### scr.grid(column=0, row=5, sticky='WE', columnspan=3)
scr.grid(column=0, row=5, columnspan=3) # sticky property removed
```

Ora, la nostra GUI ha un nuovo spazio intorno a Testo scorrevole widget, sia a sinistra che a destra. Perché abbiamo usato ilampiezza colonna=3 proprietà, la nostra Testo scorrevole widget si estende ancora su tutte e tre le colonne:



Se rimuoviamo ampiezza colonna=3, otterremo la seguente GUI, che non è ciò che vogliamo. Ora, il nostroTesto scorrevole occupa solo la colonna 0 e, a causa delle sue dimensioni, allunga il layout:

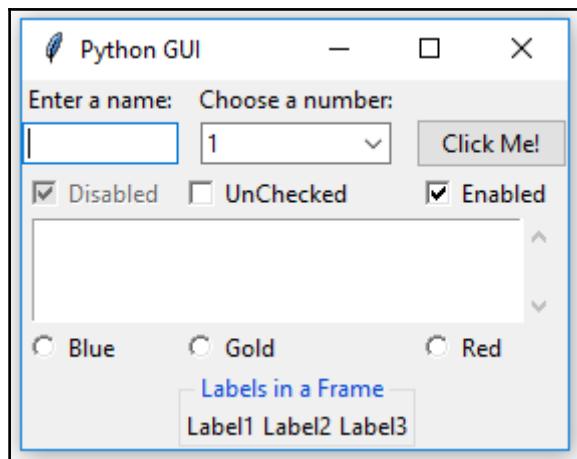
GUI_remove_columnspan.py



Un modo per riportare il nostro layout al punto in cui eravamo prima di aggiungere EtichettaFrame è quello di regolare la posizione della colonna della griglia. Modifica il valore della colonna da 0 per 1:

```
buttons_frame.grid(column=1, row=7) # now in col 1
```

Ora la nostra GUI ha il seguente aspetto:



Come funziona...

Poiché stiamo ancora utilizzando widget individuali, il nostro layout può essere incasinato. Spostando il valore della colonna di EtichettaFrame a partire dal 0 per 1, siamo stati in grado di riportare i controlli dove erano prima e dove preferiamo che fossero. Almeno, l'etichetta più a sinistra, il testo, Pulsante di spunta, ScrolledText, e Pulsante radio i widget si trovano ora dove noi intendeva che fossero. La seconda etichetta e testoIscrizione situato in colonna 1 allineati al centro della lunghezza del **Etichette in una cornice** widget, quindi abbiamo sostanzialmente spostato la nostra sfida di allineamento di una colonna a destra. Non è così visibile perché la dimensione del **Scegli un numero**: l'etichetta è quasi uguale alla dimensione del **Etichette in una cornice** titolo, e quindi la larghezza della colonna era già vicina alla nuova larghezza generata da EtichettaFrame.

C'è più...

Nella prossima ricetta, *Allineare i widget della GUI incorporando frame all'interno di frame*, incorporeremo i frame all'interno dei frame per evitare il disallineamento accidentale dei widget che abbiamo appena sperimentato in questa ricetta.

Allineare i widget della GUI incorporando i frame all'interno dei frame

Avremo un controllo molto migliore del nostro layout della GUI se incorporiamo frame all'interno di frame. Questo è quello che faremo in questa ricetta.

Prepararsi

Il comportamento dinamico di Python e dei suoi moduli GUI può creare una piccola sfida per ottenere davvero la nostra GUI come vogliamo. Qui, incorporeremo i frame all'interno dei frame per ottenere un maggiore controllo del nostro layout. Ciò stabilirà una gerarchia più forte tra i diversi elementi dell'interfaccia utente, rendendo l'aspetto visivo più facile da ottenere.

Continueremo a utilizzare la GUI che abbiamo creato nella ricetta precedente, *Come i widget espandono dinamicamente la GUI*.

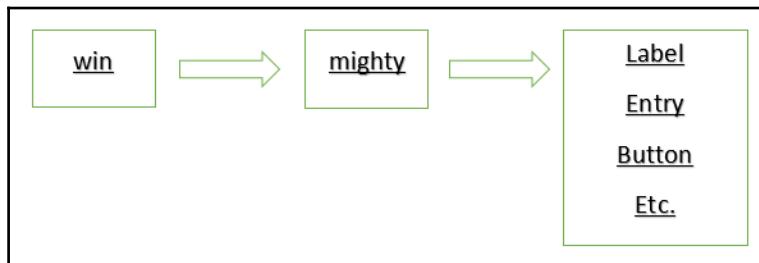
Come farlo...

Qui creeremo un frame di primo livello che conterrà altri frame e widget. Questo ci aiuterà a ottenere il layout della nostra GUI proprio come vogliamo.

Per fare ciò, dovremo incorporare i nostri controlli attuali all'interno di un frame centrale chiamato `ttk.LabelFrame`. questo telaio `ttk.LabelFrame` è il figlio della finestra principale principale e tutti i controlli saranno i figli di questa `ttk.LabelFrame`.

Fino a questo punto nelle nostre ricette, avevamo assegnato direttamente tutti i widget al nostro frame principale della GUI. Ora, assegneremo solo `EtichettaFrame` alla nostra finestra principale e dopo, lo faremo `EtichettaFrame` il contenitore padre per tutti i widget.

Questo crea la seguente gerarchia nel nostro layout della GUI:



Nel diagramma precedente, **vincere** è la variabile che contiene un riferimento al frame della finestra tkinter della nostra GUI principale, **potente** è la variabile che contiene un riferimento al nostro EtichettaFrame ed è un figlio del frame della finestra principale (**vincere**), e **Etichetta** e tutti gli altri widget sono ora inseriti nel intor EtichettaFrame contenitore (**potente**).

Aggiungi il seguente codice nella parte superiore del nostro modulo Python:

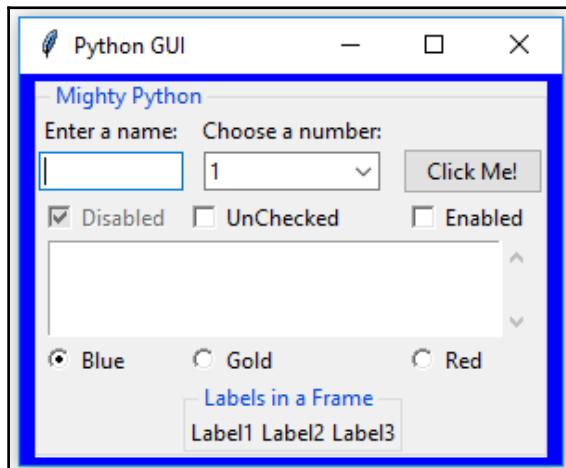
GUI_embed_frames.py

```
6#=====
7# imports
8#=====
9import tkinter as tk
10from tkinter import ttk
11from tkinter import scrolledtext
12
13# Create instance
14win = tk.Tk()
15
16# Add a title
17win.title("Python GUI")
18
19# We are creating a container frame to hold all other widgets
20mighty = ttk.LabelFrame(win, text=' Mighty Python ')
21mighty.grid(column=0, row=0, padx=8, pady=4)
```

Successivamente, modificheremo i seguenti controlli da utilizzare come genitore, sostituendo vincere. Ecco un esempio di come farlo:

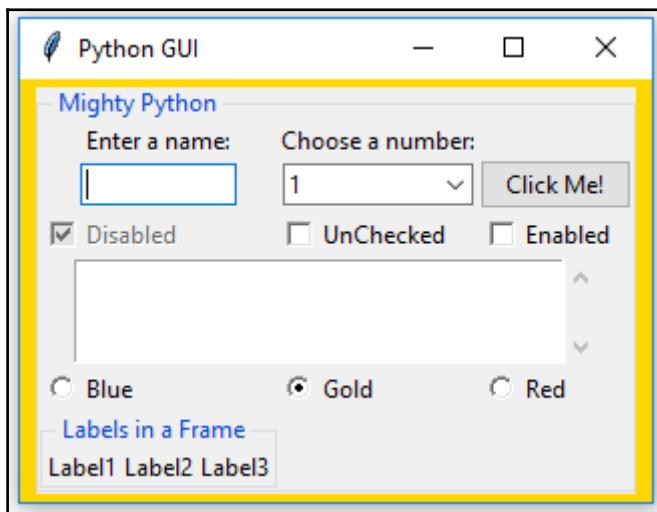
```
23 # Modify adding a Label using mighty as the parent instead of win
24 a_label = ttk.Label(mighty, text="Enter a name:")
25 a_label.grid(column=0, row=0)
```

Ciò si traduce nella seguente GUI:



Nota come tutti i widget sono ora contenuti nel **Possente Pitone** EtichettaFrame che li circonda tutti con una linea sottile appena visibile. Successivamente, possiamo ripristinare il **Etichette in una cornice** widget a sinistra senza rovinare il nostro layout della GUI:

GUI_embed_frames_align.py



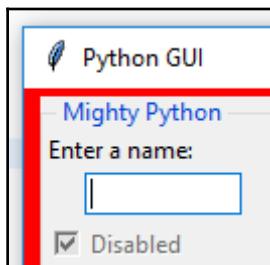
Ops - forse no. Mentre la nostra cornice all'interno di un'altra cornice si allineava bene a sinistra, ha nuovamente spostato i nostri widget in alto al centro (impostazione predefinita).

Per allinearli a sinistra, dobbiamo forzare il layout della nostra GUI utilizzando il tasto appiccicoso proprietà. Assegnandolo 'W' (West), possiamo controllare che il widget sia allineato a sinistra:

GUI_embed_frames_align_west.py

```
# Modify adding a Label using mighty as the parent instead of win
a_label = ttk.Label(mighty, text="Enter a name:")
a_label.grid(column=0, row=0, sticky='W')
```

Ciò si traduce nella seguente GUI:



Come funziona...

Nota come abbiamo allineato l'etichetta, ma non la casella di testo sottostante. Dobbiamo usare il `appiccicoso` proprietà per tutti i controlli che vogliamo allineare a sinistra. Possiamo farlo in un ciclo, usando il `winfo_children()` e `grid_configure(sticky='W')` proprietà, come abbiamo fatto prima in la ricetta, *Utilizzo della spaziatura interna per aggiungere spazio intorno ai widget*, in questo capitolo.

Il `winfo_children()` la funzione restituisce un elenco di tutti i figli appartenenti al genitore. Questo ci consente di scorrere tutti i widget e di modificarne le proprietà.

- L'uso di tkinter per forzare la denominazione a sinistra, a destra, in alto o in basso è molto simile a Java: West, East, North e South, abbreviato in 'W' e così via. Possiamo anche usare la seguente sintassi: `tk.W` invece di 'W'. Ciò richiede di aver importato il tkinter modulo alias come tk.
- In una ricetta precedente, abbiamo unito sia 'W' e 'E' per fare il nostro Testo scorrevole widget si attacca sia al lato sinistro che a quello destro del suo contenitore, usando 'NOI'. Possiamo aggiungere più combinazioni: 'NSE' estenderà il nostro widget in alto, in basso ea destra. Se abbiamo solo un widget nel nostro modulo, ad esempio un pulsante, possiamo farlo riempire l'intero frame utilizzando tutte le opzioni: 'NSWE'. Possiamo anche usare la sintassi della tupla: `appiccicoso=(tk.N, tk.S, tk.W, tk.E)`.

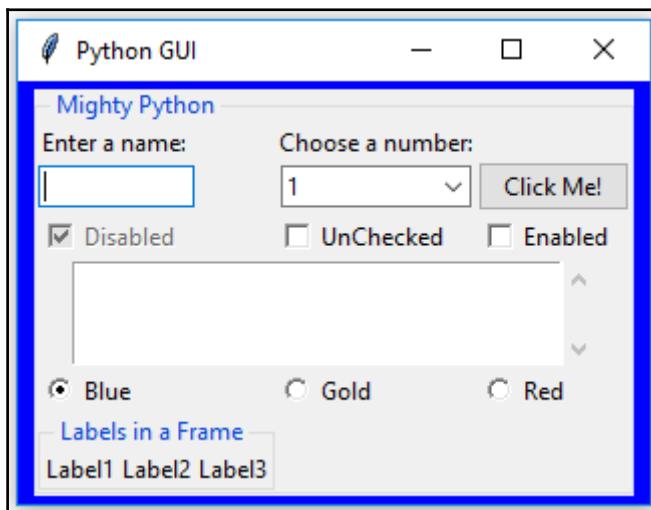


Allineiamo la voce in colonna 0 A sinistra:

```
# Adding a Textbox Entry widget
name = tk.StringVar()
name_entered = ttk.Entry(mighty, width=12, textvariable=name)
name_entered.grid(column=0, row=1, sticky=tk.W) # align left/West
```

Ora, sia l'etichetta che la voce sono allineate verso Ovest (a sinistra):

GUI_embed_frames_align_entry_west.py



Al fine di separare l'influenza che la lunghezza del nostro **Etichette in una cornice** EtichettaFrame ha sul resto del layout della nostra GUI, non dobbiamo posizionarlo EtichettaFrame nello stesso EtichettaFrame come gli altri widget ma assegnalo direttamente al modulo GUI principale (vincere). Lo faremo nei capitoli successivi.

Creazione di barre dei menu

In questa ricetta, aggiungeremo una barra dei menu alla nostra finestra principale, aggiungeremo menu alla barra dei menu e quindi aggiungeremo voci di menu ai menu.

Prepararsi

Inizieremo imparando le tecniche su come aggiungere una barra dei menu, diversi menu e alcune voci di menu per mostrare il principio di come farlo. All'inizio, fare clic su una voce di menu non avrà alcun effetto. Aggiungeremo quindi funzionalità alle voci di menu, ad esempio chiudendo la finestra principale quando si fa clic sul pulsante **Uscita** voce di menu e visualizzazione di a **Aiuto | Di** dialogo.

Continueremo ad estendere la GUI che abbiamo creato nella ricetta precedente, *Allineamento dei widget della GUI incorporando frame all'interno di frame*.

Come farlo...

Per prima cosa, dovremo importare il Menù classe da tkinter. Aggiungi la seguente riga di codice all'inizio del modulo Python, dove risiedono le istruzioni di importazione:

```
6# =====
7# imports
8# =====
9import tkinter as tk
10from tkinter import ttk
11from tkinter import scrolledtext
12from tkinter import Menu
13
14# Create instance
15win = tk.Tk()
```

Successivamente, creeremo la barra dei menu. Aggiungi il seguente codice verso la parte inferiore del modulo, appena sopra dove creiamo il ciclo di eventi principale:

GUI_menuBar_tearoff.py

```
118 # Creating a Menu Bar
119 menu_bar = Menu(win)
120 win.config(menu=menu_bar)
121
122 # Create menu and add menu items
123 file_menu = Menu(menu_bar)                      # create File menu
124 file_menu.add_command(label="New")               # add File menu item
```

Nella riga 119, chiamiamo il costruttore dell'importato Menù module class e passa nella nostra istanza GUI principale, vincere. Salviamo un'istanza di Menù oggetto nel menu_bar variabile. Nella riga 120, configuriamo la nostra GUI per utilizzare il appena creatoMenù come la menù per la nostra GUI.

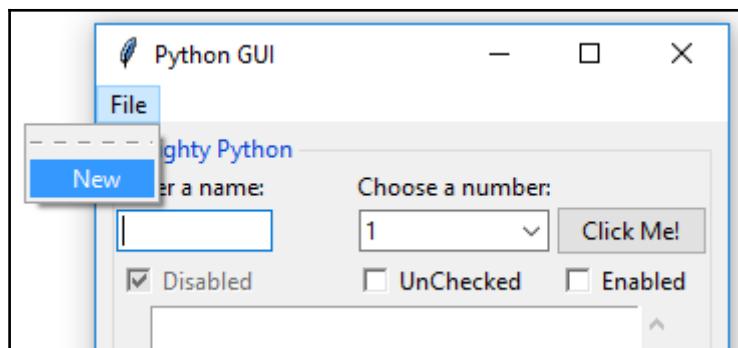
Per farlo funzionare, dobbiamo anche aggiungere il menu alla barra dei menu e dargli un'etichetta.

La voce di menu era già stata aggiunta al menu, ma dobbiamo ancora aggiungere il menu alla barra dei menu:

GUI_menubar_file.py

```
122 # Create menu and add menu items
123 file_menu = Menu(menu_bar)                      # create File menu
124 file_menu.add_command(label="New")               # add File menu item
125 menu_bar.add_cascade(label="File", menu=file_menu) # add File menu to menu bar and give it a label
126
127 name_entered.focus()      # Place cursor into name Entry
128 #=====
129 # Start GUI
130 =====
131 win.mainloop()
```

L'esecuzione di questo codice aggiunge una barra dei menu con un menu che ha una voce di menu:



Se questo tkinter barra dei menu la sintassi sembra un po' confusa, non preoccuparti.
Questa è solo la sintassi ditkinter per creare un barra dei menu. Non è molto pitone.

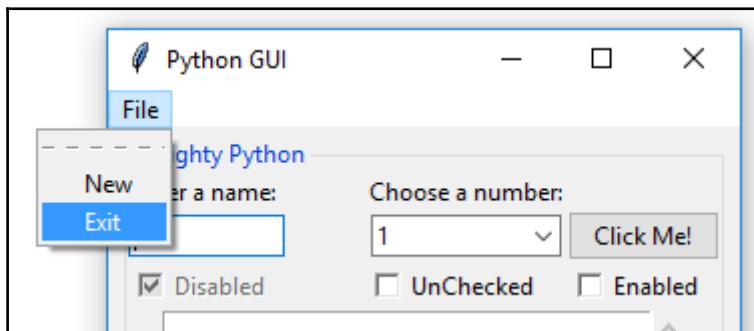


Successivamente, aggiungeremo una seconda voce di menu al primo menu che abbiamo aggiunto alla barra dei menu:

GUI_menubar_exit.py

```
122 # Add menu items
123 file_menu = Menu(menu_bar)
124 file_menu.add_command(label="New")
125 file_menu.add_command(label="Exit")
126 menu_bar.add_cascade(label="File", menu=file_menu)
127
128 name_entered.focus()      # Place cursor into name Entry
129#=====
130 # Start GUI
131 =====
132 win.mainloop()
```

L'esecuzione del codice produce il seguente risultato:



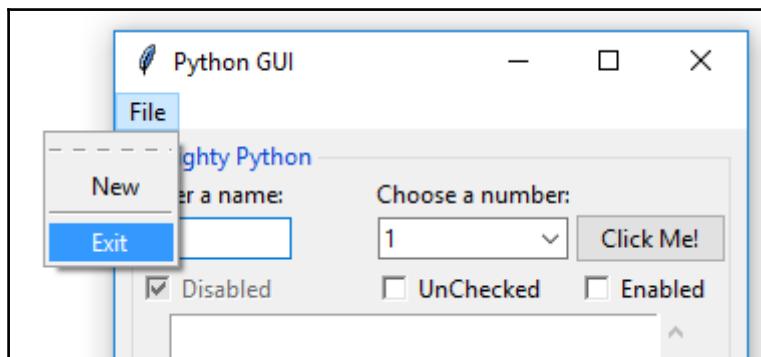
Possiamo aggiungere una linea di separazione tra le voci di menu aggiungendo questa riga di codice tra le voci di menu esistenti:

GUI_menubar_separator.py

```
# Creating a Menu Bar
menu_bar = Menu(win)
win.config(menu=menu_bar)

# Add menu items
file_menu = Menu(menu_bar)
file_menu.add_command(label="New")
file_menu.add_separator()
file_menu.add_command(label="Exit")
menu_bar.add_cascade(label="File", menu=file_menu)
```

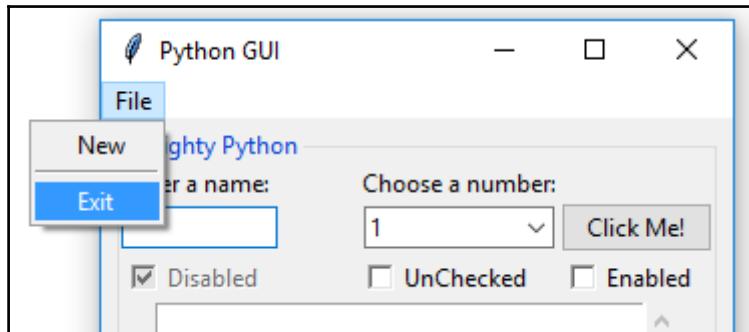
Ora possiamo vedere una linea di separazione tra le nostre due voci di menu:



Passando nel strappo proprietà al costruttore del menu, possiamo rimuovere la prima linea tratteggiata che, per impostazione predefinita, appare sopra la prima voce di menu in un menu:

```
# Add menu items
file_menu = Menu(menu_bar, tearoff=0)
file_menu.add_command(label="New")
file_menu.add_separator()
```

Ora, la linea tratteggiata non appare più e la nostra GUI ha un aspetto molto migliore:



Successivamente, aggiungeremo un secondo menu, che verrà posizionato orizzontalmente a destra del primo menu. Gli daremo una voce di menu, che chiameremo **Di** e, affinché funzioni, dobbiamo aggiungere questo secondo menu alla barra dei menu.

File e Aiuto | Disono layout di GUI di Windows molto comuni con cui tutti abbiamo familiarità e possiamo creare quegli stessi menu usando Python e tkinter:

GUI_menuBar_help.py

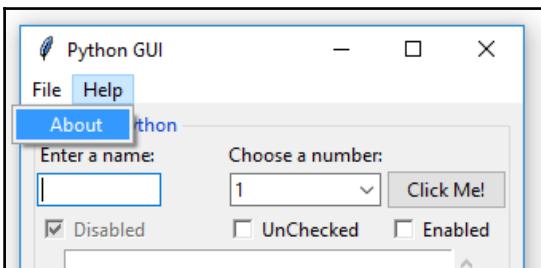
```
# Creating a Menu Bar
menu_bar = Menu(win)
win.config(menu=menu_bar)

# Add menu items
file_menu = Menu(menu_bar, tearoff=0)
file_menu.add_command(label="New")
file_menu.add_separator()
file_menu.add_command(label="Exit")
menu_bar.add_cascade(label="File", menu=file_menu)

# Add another Menu to the Menu Bar and an item
help_menu = Menu(menu_bar, tearoff=0)
menu_bar.add_cascade(label="Help", menu=help_menu)
help_menu.add_command(label="About")

name_entered.focus()      # Place cursor into name Entry
#=====
# Start GUI
#=====
win.mainloop()
```

Ora abbiamo un secondo menu con una voce di menu nella barra dei menu:



A questo punto, la nostra GUI ha una barra dei menu e due menu che contengono alcune voci di menu. Fare clic su di essi non fa molto finché non aggiungiamo alcuni comandi. Questo è quello che faremo dopo.

Aggiungi il seguente codice sopra la creazione della barra dei menu:

```
# Exit GUI cleanly
def _quit():
    win.quit()
    win.destroy()
    exit()
```

Successivamente, legheremo il **File | Uscita** voce di menu a questa funzione aggiungendo il seguente comando alla voce di menu:

GUI_menuBar_exit_quit.py

```
# Exit GUI cleanly
def _quit():
    win.quit()
    win.destroy()
    exit()

# Creating a Menu Bar
menu_bar = Menu(win)
win.config(menu=menu_bar)

# Add menu items
file_menu = Menu(menu_bar, tearoff=0)
file_menu.add_command(label="New")
file_menu.add_separator()
file_menu.add_command(label="Exit", command=_quit)
menu_bar.add_cascade(label="File", menu=file_menu)
```

Ora, quando clicchiamo su Uscita voce di menu, la nostra applicazione verrà effettivamente chiusa.

Come funziona...

Per prima cosa, chiamiamo tkinter costruttore del Menù classe. Quindi, assegniamo il menu appena creato alla nostra finestra principale della GUI. Questa, infatti, diventa la barra dei menu.

Salviamo un riferimento ad esso nella variabile di istanza denominata barra_menu.

Successivamente, creiamo un menu e vi aggiungiamo una voce di menu. Quindi aggiungiamo una seconda voce di menu al menu. Il add_cascade() allinea le voci di menu una sotto l'altra, in un layout verticale.

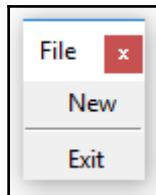
Quindi, aggiungiamo una linea di separazione tra le due voci di menu. Viene generalmente utilizzato per raggruppare voci di menu correlate e separarle da voci meno correlate (da cui il nome).

Infine, disabilitiamo il strappo linea tratteggiata per migliorare l'aspetto del nostro menu.



Senza disabilitare questa funzione predefinita, l'utente può staccare il menu dalla finestra principale. Trovo questa capacità di poco valore. Sentiti libero di giocarci facendo doppio clic sulla linea tratteggiata (prima di disabilitare questa funzione). Se stai utilizzando un Mac, questa funzione potrebbe non essere abilitata, quindi non devi preoccuparti affatto.

Controlla la seguente GUI:



Quindi aggiungiamo un secondo menu alla barra dei menu. Possiamo continuare ad aggiungere menu attraverso questa tecnica.

Successivamente, creiamo una funzione per chiudere in modo pulito la nostra applicazione GUI. Questo è il modo consigliato da Python per terminare il ciclo dell'evento principale.

Leghiamo la funzione che abbiamo creato alla voce di menu, usando i tkinter's comando proprietà. Ogni volta che vogliamo che le nostre voci di menu facciano effettivamente qualcosa, dobbiamo associare ciascuna di esse a una funzione.



Stiamo usando una convenzione di denominazione Python consigliata precedendo il nostro smettere funzione con un solo carattere di sottolineatura, per indicare che si tratta di una funzione privata che non deve essere richiamata dai client del nostro codice.

C'è più...

Aggiungeremo il **Aiuto | Difunzionalità** in Capitolo 3, *Personalizzazione aspetto e sensazione*, che introduce caselle di messaggio e molto altro.

Creazione di widget a schede

In questa ricetta, creeremo widget a schede per organizzare ulteriormente la nostra GUI in espansione scritta in tkinter.

Prepararsi

Per migliorare la nostra GUI Python utilizzando le schede, inizieremo dall'inizio, utilizzando la quantità minima di codice necessaria. In questa ricetta, creeremo una semplice GUI e quindi aggiungeremo i widget dalle ricette precedenti e li collicheremo in questo nuovo layout a schede.

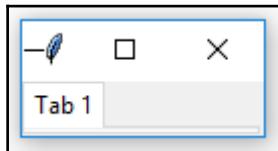
Come farlo...

Crea un nuovo modulo Python e inserisci il seguente codice in questo modulo:

GUI_tabbed.py

```
6#=====
7# imports
8#=====
9import tkinter as tk
10from tkinter import ttk
11
12win = tk.Tk()                      # Create instance
13win.title("Python GUI")             # Add a title
14tabControl = ttk.Notebook(win)       # Create Tab Control
15tab1 = ttk.Frame(tabControl)         # Create a tab
16tabControl.add(tab1, text='Tab 1')   # Add the tab
17tabControl.pack(expand=1, fill="both")# Pack to make visible
18
19#=====
20# Start GUI
21#=====
22win.mainloop()
```

Questo crea la seguente GUI:



Sebbene non sia ancora sorprendentemente impressionante, questo widget aggiunge un altro strumento molto potente al nostro toolkit di progettazione della GUI. Viene fornito con i suoi limiti nell'esempio minimalistico sopra (ad esempio, non possiamo né riposizionare la GUI né mostrare l'intero titolo della GUI).

Mentre abbiamo usato il gestore di layout della griglia per GUI più semplici nelle ricette precedenti, possiamo usare un gestore di layout più semplice e pacchetto è uno di loro.

Nel codice precedente, noi pacchetto il tabControl e ttk.Notebook widget nel modulo GUI principale, espandendo il controllo a schede del notebook per riempire tutti i lati.

Possiamo aggiungere una seconda scheda al nostro controllo e fare clic tra di loro:

GUI_tabbed_two.py

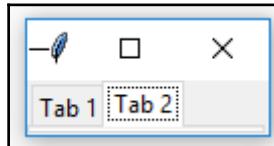
```
tabControl = ttk.Notebook(win)           # Create Tab Control

tab1 = ttk.Frame(tabControl)             # Create a tab
tabControl.add(tab1, text='Tab 1')       # Add the tab
tab2 = ttk.Frame(tabControl)             # Add a second tab
tabControl.add(tab2, text='Tab 2')        # Add second tab

tabControl.pack(expand=1, fill="both")   # Pack to make visible

#=====
# Start GUI
#=====
win.mainloop()
```

Ora abbiamo due schede. Clicca su**Scheda 2** per metterlo a fuoco:



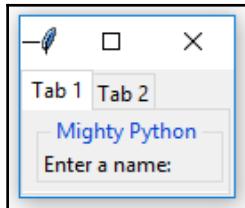
Ci piacerebbe davvero vedere il nostro titolo di Windows; quindi, per fare ciò, dobbiamo aggiungere un widget a una delle nostre schede. Il widget deve essere abbastanza largo da espandere la nostra GUI in modo dinamico per visualizzare il titolo della nostra finestra:

GUI_tabbed_two_mighty.py

```
# LabelFrame using tab1 as the parent
mighty = ttk.LabelFrame(tab1, text=' Mighty Python ')
mighty.grid(column=0, row=0, padx=8, pady=4)

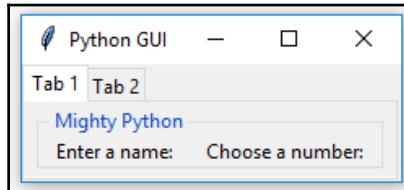
# Label using mighty as the parent
a_label = ttk.Label(mighty, text="Enter a name:")
a_label.grid(column=0, row=0, sticky='W')
```

Ora abbiamo il nostro **Possente Pitone** dentro **Scheda1**. Questo espande la nostra GUI, ma i widget aggiunti non sono abbastanza grandi da rendere visibile il titolo della GUI:



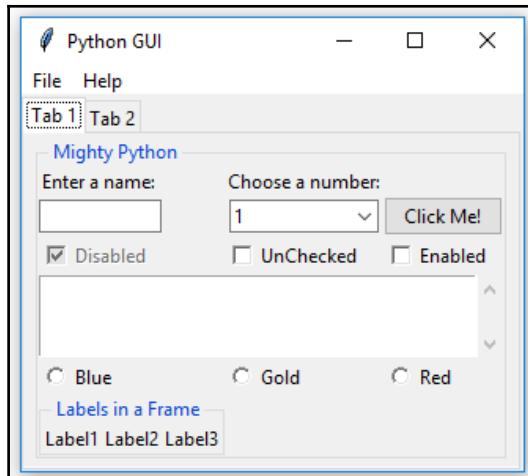
Dopo aver aggiunto una seconda etichetta più un po' di spazio attorno ad esse, allunghiamo il layout abbastanza da poter vedere di nuovo il nostro titolo della GUI:

GUI_tabbed_two_mighty_labels.py



Possiamo continuare a posizionare tutti i widget che abbiamo creato finora nei nostri controlli a schede appena creati:

GUI_tabbed_all_widgets.py



Ora, tutti i widget risiedono all'interno **Scheda 1**. Spostiamone un po' **Scheda 2**. Primo, creiamo un secondo EtichettaFrame essere il contenitore dei nostri widget che si trasferiscono in **Scheda 2**:

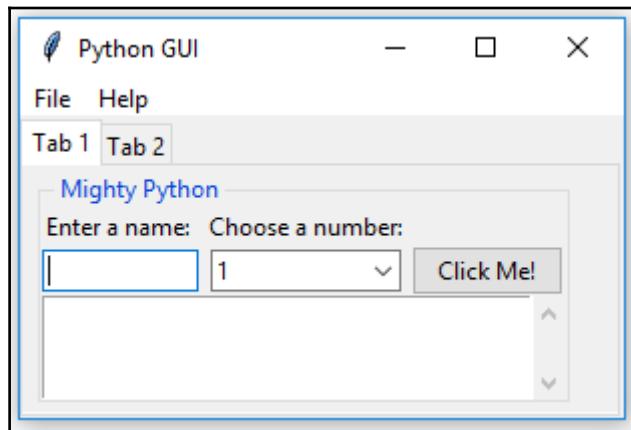
```
mighty2 = ttk.LabelFrame(tab2, text=' The Snake ')
mighty2.grid(column=0, row=0, padx=8, pady=4)
```

Successivamente, spostiamo i pulsanti Check e Radiobutton su **Scheda 2**, specificando il nuovo contenitore genitore, che è una nuova variabile che chiamiamo potente2. Ecco un esempio che applichiamo a tutti i controlli che si trasferiscono a **Scheda 2**:

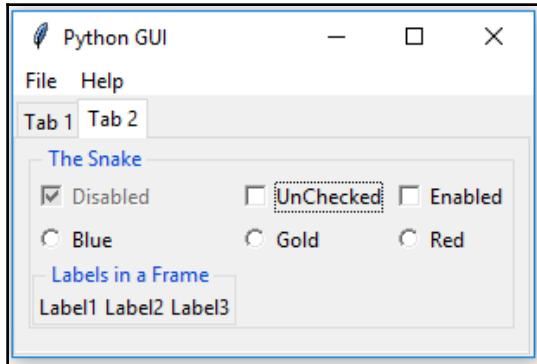
```
chVarDis = tk.IntVar()
check1 = tk.Checkbutton(mighty2, text="Disabled", variable=chVarDis, state='disabled')
```

Quando eseguiamo il codice, la nostra GUI ora ha un aspetto diverso. **Scheda 1** ha meno widget di prima, quando conteneva tutti i nostri widget creati in precedenza:

GUI_tabbed_all_widgets_both_tabs.py



Ora possiamo fare clic su **Scheda 2** e guarda i nostri widget trasferiti:



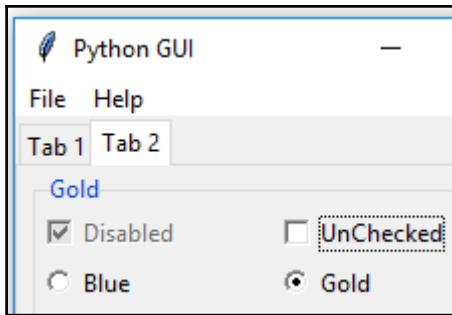
Facendo clic sul trasferimento Pulsante radio non ha più alcun effetto, quindi cambieremo le loro azioni per rinominare la proprietà text, dal titolo del EtichettaFrame widget, al nome the Tasti della radio Schermo. Quando clicchiamo su **Oro** Pulsante radio, non impostiamo più lo sfondo della cornice al colore oro ma qui sostituiamo il EtichettaFrame titolo del testo invece. Pitone **Il serpente** ora diventa **Oro**.

```
def radCall():
    radSel=radVar.get()
    if radSel == 0: mighty2.configure(text='Blue')
    elif radSel == 1: mighty2.configure(text='Gold')
    elif radSel == 2: mighty2.configure(text='Red')
```

Ora, selezionando uno dei Pulsante radio i widget cambieranno il nome del EtichettaTelaio:

GUI_tabbed_all_widgets_both_tabs_radio.py





Come funziona...

Dopo aver creato una seconda scheda, abbiamo spostato alcuni dei widget che originariamente risiedevano in **Scheda 1** per **Scheda 2**. L'aggiunta di schede è un altro modo eccellente per organizzare la nostra GUI in continua crescita. Questo è un modo molto carino per gestire la complessità nel nostro design della GUI. Possiamo organizzare i widget in gruppi, a cui appartengono naturalmente e liberare i nostri utenti dal disordine utilizzando le schede.



In tkinter, la creazione delle schede viene eseguita tramite il Taccuino widget, che è lo strumento che ci permette di aggiungere controlli a schede. Il taccuino tkinter widget, come tanti altri widget, è dotato di proprietà aggiuntive che possiamo usare e configurare. Un ottimo punto di partenza per esplorare le funzionalità aggiuntive dei widget tkinter a nostra disposizione è il sito Web ufficiale:<https://docs.python.org/3.1/library/tkinter.ttk.html#notebook>.

Utilizzo del gestore del layout della griglia

Il gestore del layout della griglia è uno degli strumenti di layout più utili a nostra disposizione. Lo abbiamo già usato in molte ricette perché è così potente.

Prepararsi...

In questa ricetta, esamineremo alcune delle tecniche del gestore del layout della griglia. Li abbiamo già utilizzati e qui li esploreremo ulteriormente.

Come farlo...

In questo capitolo, abbiamo creato righe e colonne, che è veramente un approccio di database alla progettazione della GUI (MS Excel fa lo stesso). Abbiamo codificato le prime righe, ma poi ci siamo dimenticati di fornire alla riga successiva una specifica di dove desideriamo che risieda.

Tkinter ha compilato questo per noi senza che ce ne accorgessimo.

Ecco cosa abbiamo fatto nel nostro codice:

```
# Using a scrolled Text control
scrol_w = 30
scrol_h = 3
scr = scrolledtext.ScrolledText(mighty, width=scrol_w, height=scrol_h, wrap=tk.WORD)
# scr.grid(column=0, row=2, sticky='WE', columnspan=3)
scr.grid(column=0, sticky='WE', columnspan=3)                                # row not specified
```

Tkinter aggiunge automaticamente la riga mancante dove non abbiamo specificato alcuna riga in particolare. Potremmo non rendercene conto.

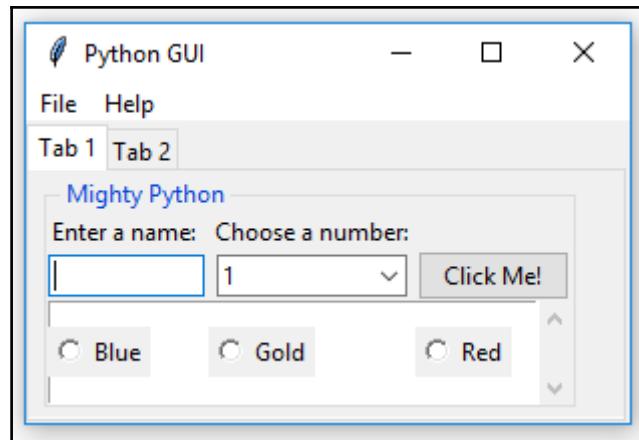
Abbiamo disposto il Iscrizione widget sulla riga 1, poi abbiamo dimenticato di specificare la riga per il nostro Testo scorrevole widget a cui facciamo riferimento tramite il scr variabile, e poi abbiamo aggiunto il Pulsante radio widget da disporre in fila 3.

Funziona bene perché tkinter ha incrementato automaticamente la posizione della riga per il nostro Testo scorrevole widget per utilizzare il numero di riga successivo più alto, che era riga 2.

Guardando il nostro codice e non rendendoci conto che abbiamo dimenticato di posizionare esplicitamente il nostro Testo scorrevole widget per remare 2, potremmo pensare che non ci risieda nulla.

Quindi, potremmo provare quanto segue. Se impostiamo la variabile curRad usare la riga 2, potremmo avere una spiacevole sorpresa:

GUI_grid_layout.py



Come funziona...

Nota come la nostra fila di Pulsante radio(s) è finito improvvisamente in mezzo al nostro Testo scorrevole aggeggio! Questo non è sicuramente quello che volevamo che fosse la nostra GUI!



Se ci dimentichiamo di specificare esplicitamente il numero di riga, per impostazione predefinita, tkinter utilizzerà la riga successiva disponibile.

Abbiamo anche usato il ampiezza delle colonne proprietà per assicurarci che i nostri widget non fossero limitati a una sola colonna. Ecco come ci siamo assicurati che il nostroTesto scorrevole widget si estende su tutte le colonne della nostra GUI:

```
# Using a scrolled Text control
scrol_w = 30
scrol_h = 3
scr = scrolledtext.ScrolledText(mighty, width=scrol_w, height=scrol_h, wrap=tk.WORD)
scr.grid(column=0, row=2, sticky='WE', columnspan=3) # using columnspan
```

3

Personalizzazione aspetto e sensazione

In questo capitolo, personalizzeremo la nostra GUI utilizzando Python 3.6 e versioni successive. Tratteremo le seguenti ricette:

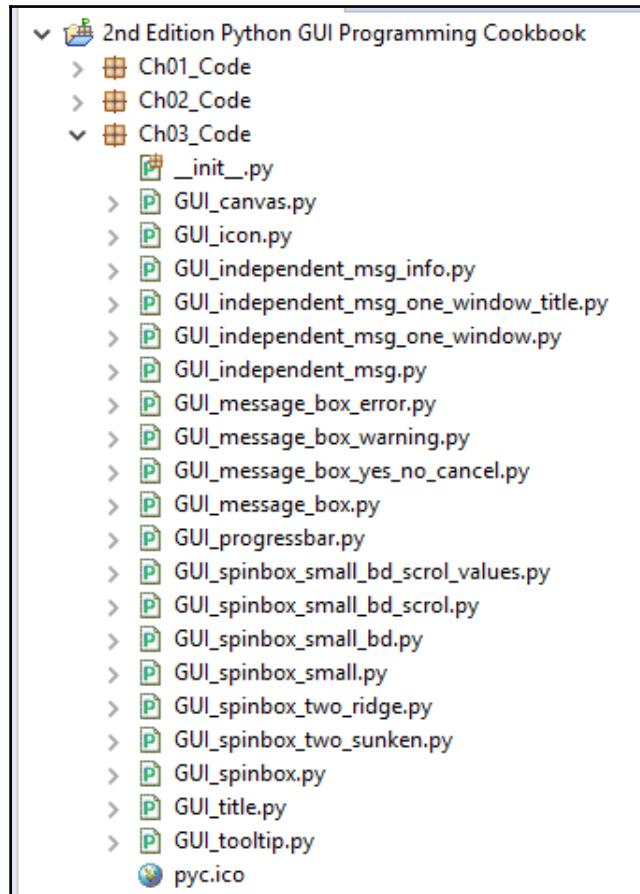
- Creazione di finestre di messaggio: informazioni, avvisi ed errori
- Come creare finestre di messaggio indipendenti
- Come creare il titolo di un modulo di finestra di tkinter Cambiare
- l'icona della finestra principale principale Usando un controllo della casella di selezione
- Aspetto in rilievo, incavato e in rilievo dei widget
- Creazione di tooltip usando Python
- Aggiunta di una barra di avanzamento alla
- GUI Come utilizzare il widget canvas

introduzione

In questo capitolo, personalizzeremo alcuni dei widget nella nostra GUI modificando alcune delle loro proprietà. Introdurremo anche alcuni nuovi widget che tkinter ci offre.

Il *Creazione di tooltip usando Python* la ricetta creerà un Descrizione comando Classe in stile OOP, che farà parte del singolo modulo Python che abbiamo utilizzato fino ad ora.

Ecco la panoramica dei moduli Python per questo capitolo:



Creazione di finestre di messaggio: informazioni, avvisi ed errori

Una finestra di messaggio è una finestra pop-up che fornisce un feedback all'utente. Può essere informativo, suggerendo potenziali problemi così come errori catastrofici.

Usare Python per creare finestre di messaggio è molto semplice.

Prepararsi

Aggiungeremo funzionalità al **Aiuto | Divoce di menu che abbiamo creato nel capitolo precedente, nel *Creazione di widget a schede* ricetta.**

Il codice è di `GUI_tabbed_all_widgets_both_tabs.py`. Il tipico feedback all'utente quando si fa clic su **Aiuto | Dimenu** nella maggior parte delle applicazioni è informativo. Iniziamo con queste informazioni e poi modifichiamo il modello di progettazione per mostrare avvisi ed errori.

Come farlo...

Aggiungi la seguente riga di codice nella parte superiore del modulo in cui risiedono le istruzioni di importazione:

```
#=====
# imports
#=====
import tkinter as tk
from tkinter import ttk
from tkinter import scrolledtext
from tkinter import Menu
from tkinter import messagebox as msg
```

Quindi, crea una funzione di callback che visualizzerà una finestra di messaggio. Dobbiamo individuare il codice della richiamata sopra il codice dove alleghiamo la richiamata alla voce di menu, perché questo è ancora procedurale e non codice OOP.

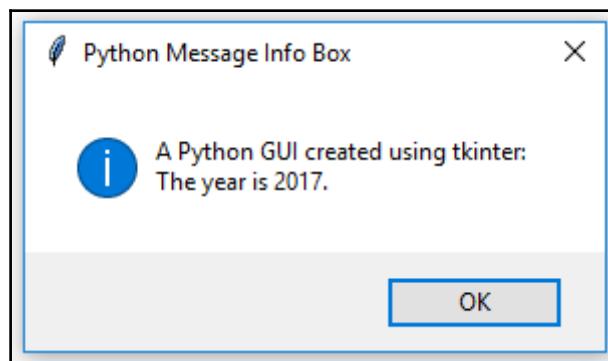
Aggiungi il seguente codice appena sopra le righe in cui creiamo il menu di aiuto:

`GUI_message_box.py`

```
# Display a Message Box
def _msgBox():
    msg.showinfo('Python Message Info Box', 'A Python GUI created using tkinter:\nThe year is 2017. ')

# Add another Menu to the Menu Bar and an item
help_menu = Menu(menu_bar, tearoff=0)
help_menu.add_command(label="About", command=_msgBox)    # display messagebox when clicked
menu_bar.add_cascade(label="Help", menu=help_menu)
```

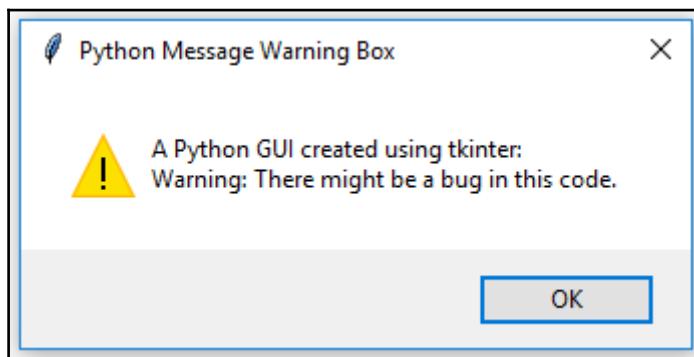
Facendo clic su **Aiuto** | **Diora** fa apparire la seguente finestra pop-up:



Trasformiamo invece questo codice in una finestra pop-up con un messaggio di avviso.
Commenta la riga precedente e aggiungi il seguente codice:

```
# Display a Message Box
def _msgBox():
#     msg.showinfo('Python Message Info Box', 'A Python GUI created using tkinter:'
#                 '\nThe year is 2017.')
    msg.showwarning('Python Message Warning Box', 'A Python GUI created using tkinter:'
                   '\nWarning: There might be a bug in this code.')
```

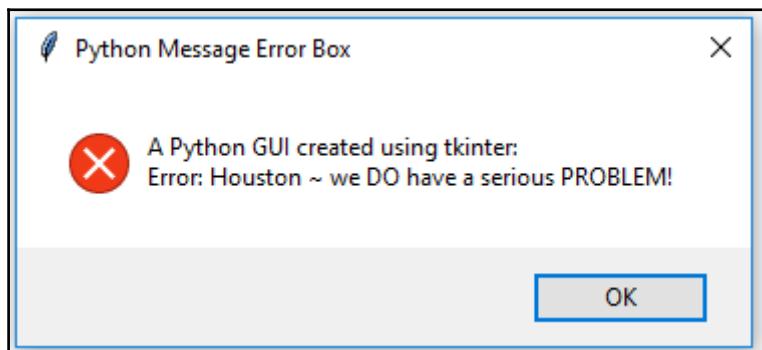
L'esecuzione del codice precedente ora risulterà nella seguente finestra di messaggio leggermente modificata:



La visualizzazione di una finestra di messaggio di errore è semplice e di solito avverte l'utente di un problema serio. Come abbiamo fatto nel frammento di codice precedente, commenta la riga precedente e aggiungi il codice seguente, come abbiamo fatto qui:

```
# Display a Message Box
def _msgBox():
#     msg.showinfo('Python Message Info Box', 'A Python GUI created using tkinter:\nThe year is 2017.')
#     msg.showwarning('Python Message Warning Box', 'A Python GUI created using tkinter:\'
#                     '\nWarning: There might be a bug in this code.')
    msg.showerror('Python Message Error Box', 'A Python GUI created using tkinter:\'
                  '\nError: Houston ~ we DO have a serious PROBLEM!')
```

Il messaggio di errore è simile a questo:



Come funziona...

Abbiamo aggiunto un'altra funzione di callback e l'abbiamo collegata come delegato per gestire l'evento click. Ora, quando clicchiamo su **Aiuto | Dimenu**, viene eseguita un'azione. Stiamo creando e visualizzando le finestre di dialogo pop-up più comuni. Sono modali, quindi l'utente non può utilizzare la GUI finché non fa clic su **ok** pulsante.

Nel primo esempio mostriamo una finestra informativa, come si vede dall'icona alla sua sinistra. Successivamente, creiamo finestre di messaggio di avviso e di errore, che cambiano automaticamente l'icona associata al popup. Tutto quello che dobbiamo fare è specificare quale finestra di messaggio vogliamo visualizzare.

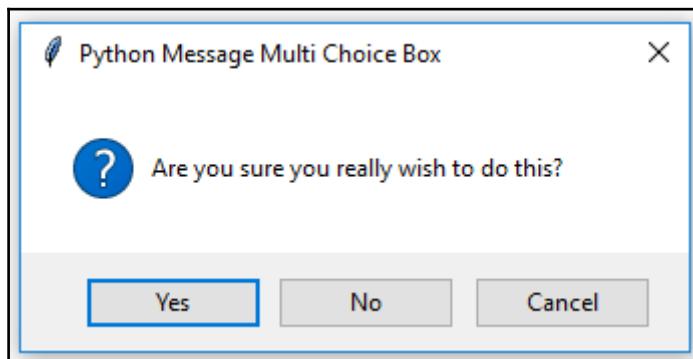
Esistono diverse finestre di messaggio che ne visualizzano più di una **ok** pulsante e possiamo programmare le nostre risposte in base alla selezione dell'utente.

Quello che segue è un semplice esempio che illustra questa tecnica:

```
# Display a Message Box
def _msgBox():
#    msg.showinfo('Python Message Info Box', 'A Python GUI created using tkinter:\nThe year is 2017.')
#    msg.showwarning('Python Message Warning Box', 'A Python GUI created using tkinter:\nWarning: There might be a bug in this code.')
#    msg.showerror('Python Message Error Box', 'A Python GUI created using tkinter:\nError: Houston ~ we DO have a serious PROBLEM!')
    answer = msg.askyesnocancel("Python Message Multi Choice Box", "Are you sure you really wish to do this?")
print(answer)
```

L'esecuzione di questo codice GUI si traduce in un popup la cui risposta dell'utente può essere utilizzata per ramificarsi sulla risposta di questo ciclo GUI guidato da eventi, salvandolo nel risposta variabile:

GUI_message_box_yes_no_cancel.py



L'output della console che utilizza Eclipse mostra che facendo clic su **sì** Il pulsante restituisce il valore booleano di Vero essere assegnato al risposta variabile:



Ad esempio, potremmo utilizzare il seguente codice:

Se risposta == Vero:
<fai qualcosa>

Facendo clic su **No** ritorna falso e **Annulla** ritorna Nessuna.

Come creare finestre di messaggio indipendenti independent

In questa ricetta, creeremo le nostre finestre di messaggio tkinter come finestre GUI di primo livello autonome.

Noterai innanzitutto che, così facendo, ci ritroveremo con una finestra in più, quindi esploreremo modi per nascondere questa finestra.

Nella ricetta precedente, abbiamo richiamato le caselle di messaggio di tkinter tramite il nostro **Aiuto | Di**menu dal nostro modulo GUI principale.

Quindi, perché dovremmo voler creare una finestra di messaggio indipendente?

Uno dei motivi è che potremmo personalizzare le nostre finestre di messaggio e riutilizzarle in molte delle nostre GUI. Invece di dover copiare e incollare lo stesso codice in ogni GUI Python che progettiamo, possiamo estrarlo dal nostro codice GUI principale. Questo può creare un piccolo componente riutilizzabile che possiamo quindi importare in diverse GUI Python.

Prepararsi

Abbiamo già creato il titolo di una finestra di messaggio nella ricetta precedente, *Creazione di finestre di messaggio: informazioni, avvisi ed errori*. Non riutilizzeremo il codice della ricetta precedente ma costruiremo una nuova GUI utilizzando pochissime righe di codice Python.

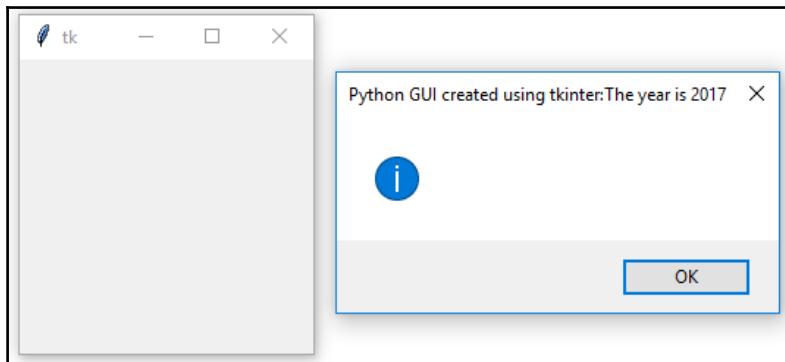
Come farlo...

Possiamo creare una semplice finestra di messaggio, come segue:

```
from tkinter import messagebox as msg  
msg.showinfo('Python GUI created using tkinter:\nThe year is 2017')
```

Ciò si tradurrà nelle seguenti due finestre:

GUI_independent_msg.py



Questo non sembra quello che avevamo in mente. Ora abbiamo due finestre, una indesiderata e la seconda con il testo visualizzato come titolo.

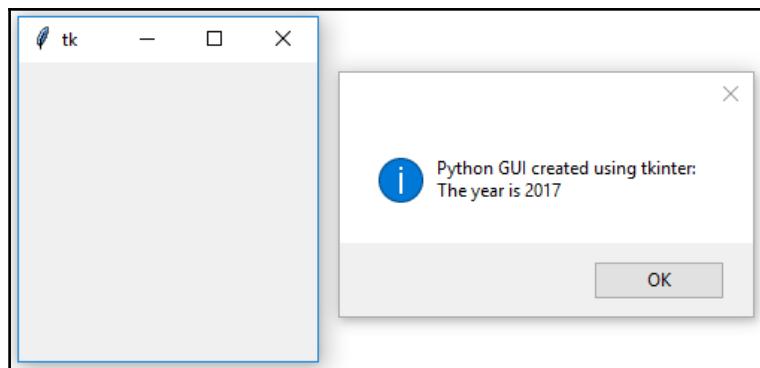
Ops!

Risolviamo questo ora. Possiamo cambiare il codice Python aggiungendo una virgoletta singola o doppia, seguita da una virgola:

```
from tkinter import messagebox as msg  
msg.showinfo('', 'Python GUI created using tkinter:\nThe year is 2017')
```

Ora, non abbiamo un titolo ma il nostro testo è finito all'interno del popup, come avevamo previsto:

GUI_independent_msg_info.py



Il primo parametro è il titolo e il secondo è il testo visualizzato nella finestra di messaggio a comparsa. Aggiungendo una coppia vuota di virgolette singole o doppie, seguite da una virgola, possiamo spostare il nostro testo dal titolo nella finestra di messaggio pop-up.

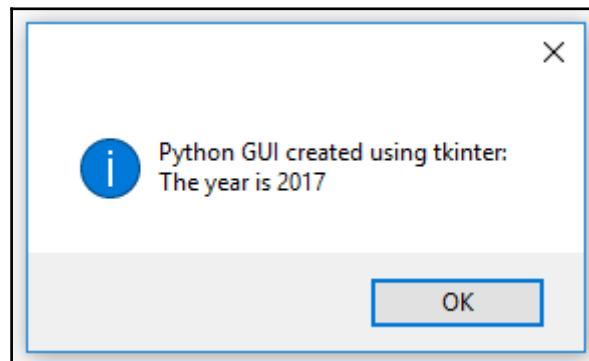
Abbiamo ancora bisogno di un titolo e vogliamo assolutamente liberarcene di questa seconda finestra non necessaria. La seconda finestra è causata da un ciclo di eventi di Windows. Possiamo liberarcene sopprimendolo.

Aggiungi il seguente codice:

```
from tkinter import messagebox as msg
from tkinter import Tk
root = Tk()
root.withdraw()
msg.showinfo('', 'Python GUI created using tkinter:\nThe year is 2017')
```

Ora abbiamo solo una finestra. Il ritirare() la funzione rimuove la finestra di debug che non ci interessa avere fluttuante:

GUI_independent_msg_one_window.py



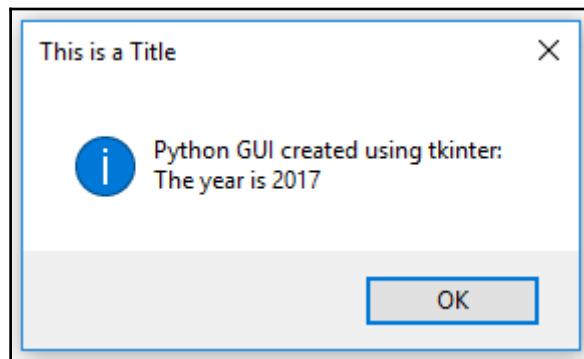
Per aggiungere un titolo, tutto ciò che dobbiamo fare è inserire una stringa nel nostro primo argomento vuoto.

Ad esempio, considera il seguente frammento di codice:

```
from tkinter import messagebox as msg
from tkinter import Tk
root = Tk()
root.withdraw()
msg.showinfo('This is a Title', 'Python GUI created using tkinter:\nThe year is 2017')
```

Ora la nostra finestra di dialogo ha un titolo, come mostrato nello screenshot seguente:

GUI_independent_msg_one_window_title.py



Come funziona...

Stiamo passando più argomenti nel tkinter costruttore della finestra di messaggio per aggiungere un titolo al form della finestra e visualizzare il testo nella finestra di messaggio invece di visualizzarlo come titolo. Ciò accade a causa della posizione degli argomenti che stiamo passando. Se tralasciamo una citazione vuota o una doppia virgoletta, il widget della finestra di messaggio prende la prima posizione degli argomenti come titolo, non il testo da visualizzare all'interno della finestra di messaggio. Passando una citazione vuota seguita da una virgola, cambiamo dove la finestra del messaggio mostra il testo che stiamo passando alla funzione.

Sopprimiamo la seconda finestra pop-up, che viene creata automaticamente dal widget della finestra di messaggio di tkinter, chiamando il pulsante ritirare() metodo sulla nostra finestra principale principale.

Come creare il titolo di un modulo finestra tkinter

Il principio di cambiare il titolo di una finestra principale di tkinter è lo stesso di quello che abbiamo discusso nella ricetta precedente: *Come creare caselle di messaggio indipendenti*. Passiamo semplicemente una stringa come primo argomento al costruttore del widget.

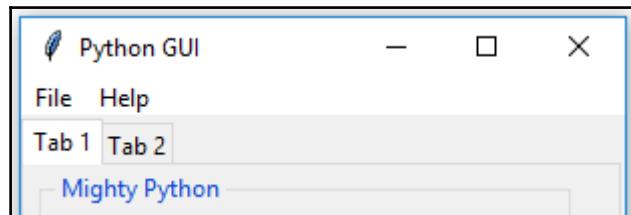
Prepararsi

Invece di una finestra di dialogo pop-up, creiamo la finestra principale principale e le diamo un titolo.

Come farlo...

Il codice seguente crea la finestra principale e vi aggiunge un titolo. Lo abbiamo già fatto nelle ricette precedenti, ad esempio in *Creazione di widget a schede*, nel Capitolo 2, *Gestione del layout*. Qui ci concentriamo solo su questo aspetto della nostra GUI:

```
importa tkinter come tk
win = tk.Tk()                      # Crea istanza
win.title ("GUI Python")            # Aggiungi un titolo
```



Come funziona...

Questo dà un titolo alla finestra principale usando il tkinter integrato titolo proprietà. Dopo aver creato unTk() esempio, possiamo usare tutto il built-in tkinter proprietà per personalizzare la nostra GUI.

Cambiare l'icona della finestra principale di root

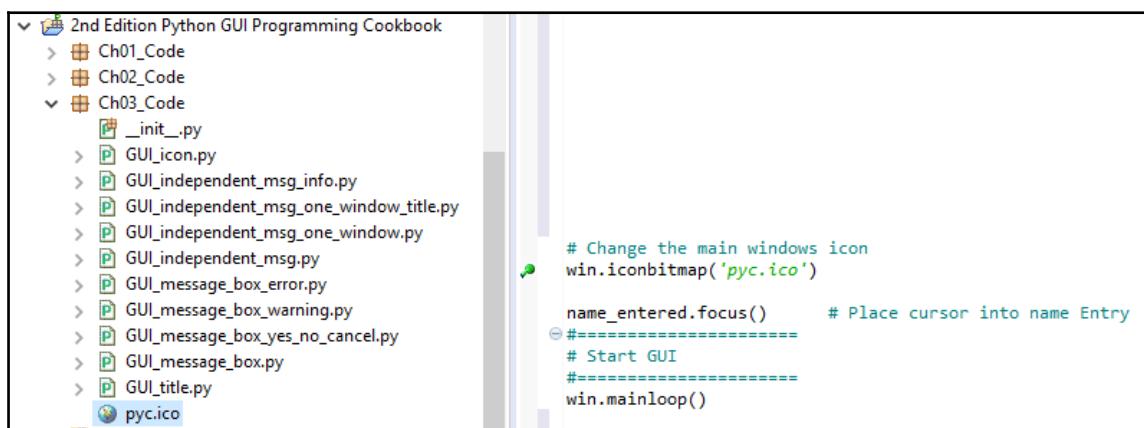
Un modo per personalizzare la nostra GUI è dargli un'icona diversa dall'icona predefinita fornita con tkinter. Ecco come lo facciamo.

Prepararsi

Stiamo migliorando la nostra GUI dalla ricetta, *Creazione di widget a schede*, nel Capitolo 2, *Gestione del layout*. Utilizzeremo un'icona fornita con Python, ma puoi utilizzare qualsiasi icona che ritieni utile. Assicurati di avere il percorso completo della posizione dell'icona nel codice, altrimenti potresti ricevere errori.

Come farlo...

Per questo esempio, ho copiato l'icona da cui ho installato Python 3.6 nella stessa cartella in cui risiede il codice.



```
# Change the main windows icon
win.iconbitmap('pyc.ico')

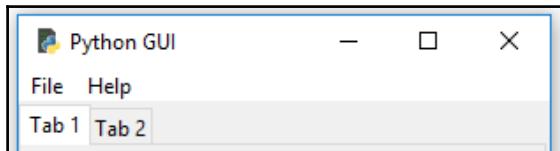
name_entered.focus()      # Place cursor into name Entry
#=====
# Start GUI
#=====
win.mainloop()
```

Inserisci il seguente codice da qualche parte sopra il ciclo dell'evento principale:

```
# Cambia l'icona di Windows principale
win.iconbitmap('pyc.ico')
```

Nota come il *piuma* l'icona predefinita nell'angolo in alto a sinistra della GUI è cambiata:

GUI_icon.py



Come funziona...

Questa è un'altra proprietà che viene spedita con tkinter, che viene fornito con Python 3.6 e versioni successive. Noi usiamo il `iconbitmap` proprietà per cambiare l'icona del nostro form della finestra principale principale, passando un percorso relativo a un'icona. Questo sovrascrive l'icona predefinita di tkinter, sostituendola con la nostra icona preferita.

Utilizzo di un controllo della casella numerica

In questa ricetta useremo a Spinbox widget, e legheremo anche il *accedere* tasto sulla tastiera a uno dei nostri widget.

Prepararsi

Useremo la nostra GUI a schede e aggiungeremo a Spinbox widget sopra il Testo scorrevole controllo. Questo richiede semplicemente di incrementare il Testo scorrevole riga il valore di uno e per inserire il nostro nuovo Spinbox controllo nella riga sopra il Iscrizione aggeggio.

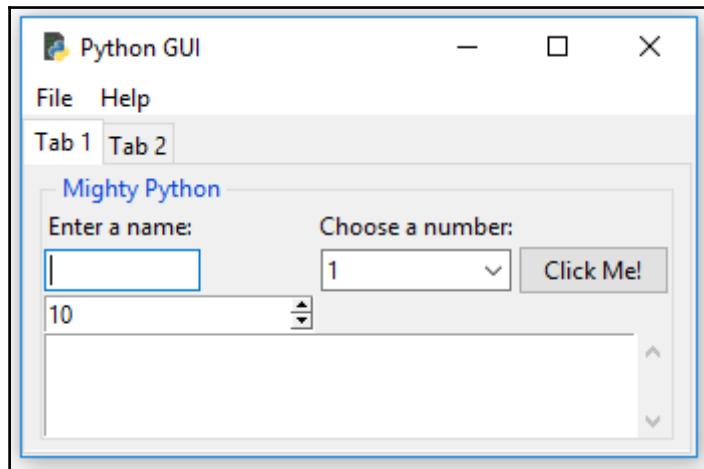
Come farlo...

Per prima cosa, aggiungiamo il Spinbox controllo. Inserisci il seguente codice sopra il Testo scorrevole aggeggio:

```
# Aggiunta di un widget Casella rotante
spin = Spinbox(mighty, from_=0, to=10)
spin.grid(column=0, row=2)
```

Questo modificherà la nostra GUI come segue:

GUI_spinbox.py

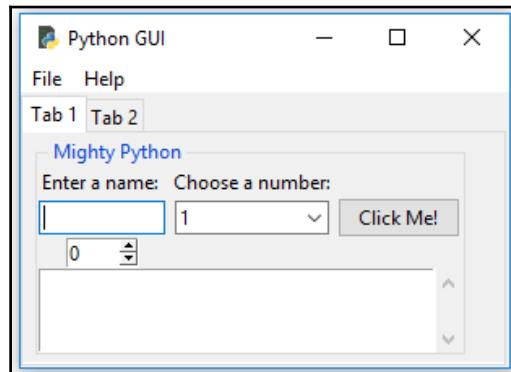


Successivamente, ridurremo la dimensione del Spinbox aggiungendo:

```
spin = Spinbox(mighty, from_=0, to=10, width=5)
```

L'esecuzione del codice precedente risulta nella seguente GUI:

GUI_spinbox_small.py

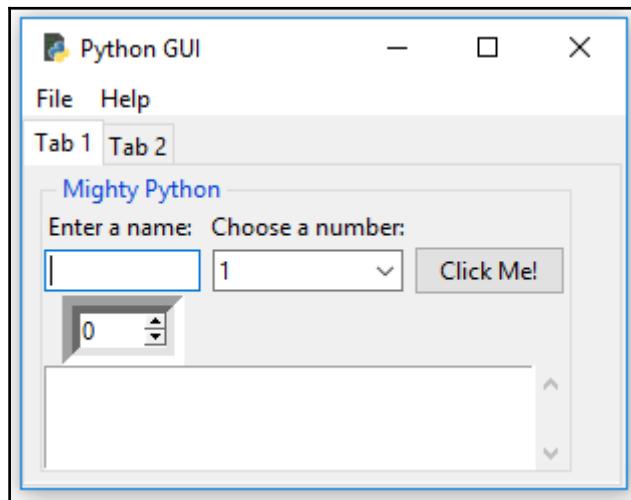


Successivamente, aggiungiamo un'altra proprietà per personalizzare ulteriormente il nostro widget; bd è una notazione abbreviata per il larghezza del bordo proprietà:

```
spin = Spinbox(mighty, from_=0, to=10, width=5 , bd=8)
```

L'esecuzione del codice precedente risulta nella seguente GUI:

GUI_spinbox_small_bd.py



Qui, aggiungiamo funzionalità al widget creando un callback e collegandolo al controllo.

Questo stamperà la selezione del Spinbox in Testo scorrevole così come su stdout.

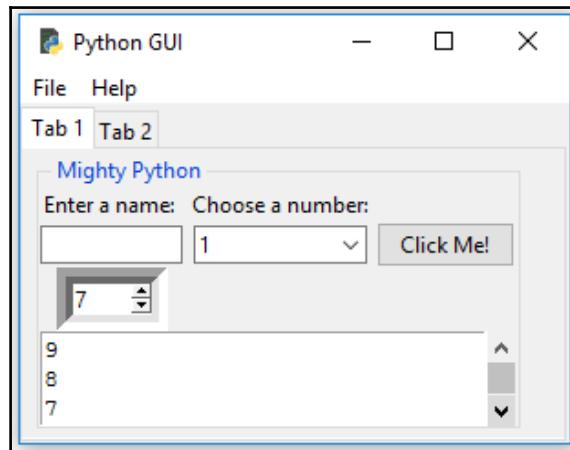
La variabile denominata scorrere è il nostro riferimento al Testo scorrevole aggeggio:

```
# Richiamata Spinbox
def _spin():
    valore = spin.get()
    stampa (valore)
    scrol.insert(tk.INSERT, valore + '\n')

spin = Spinbox(mighty, from_=0, to=10, width=5, bd=8,
               comando=_spin)
```

L'esecuzione del codice precedente risulta nella seguente GUI:

GUI_spinbox_small_bd_scrol.py

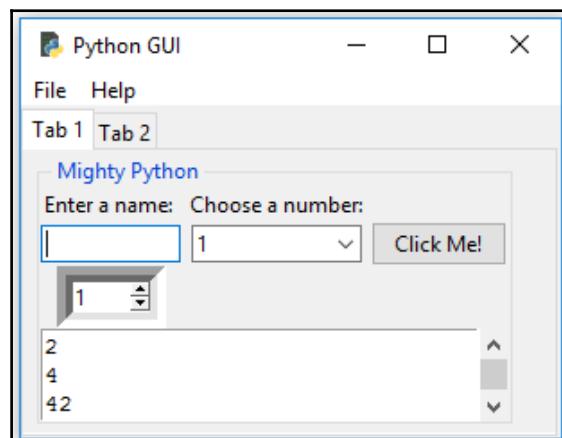


Invece di usare un intervallo, possiamo anche specificare un insieme di valori:

```
# Aggiunta di un widget Spinbox utilizzando un insieme di valori
spin = Spinbox(potente, valori = (1, 2, 4, 42, 100), larghezza = 5, bd = 8,
                comando=_spin)
spin.grid(colonna=0, riga=2)
```

Questo creerà il seguente output della GUI:

GUI_spinbox_small_bd_scrol_values.py



Come funziona...

Nota come, nel primo screenshot, il nostro nuovo Spinbox controllo predefinito a una larghezza di 20, spingendo fuori la larghezza della colonna di tutti i controlli in questa colonna. Questo non è quello che vogliamo.

Abbiamo dato al widget un intervallo da 0 a 10.

Nella seconda schermata, abbiamo ridotto la larghezza del Spinbox controllo, che lo allineava al centro della colonna.

Nel terzo screenshot, abbiamo aggiunto il larghezza del bordo proprietà del Spinbox, che ha reso automaticamente l'intero Spinbox appaiono non più piatti, ma tridimensionali.

Nella quarta schermata, abbiamo aggiunto una funzione di richiamata per visualizzare il numero scelto nel Testo scorrevole widget e stampalo sul flusso di uscita standard. Abbiamo aggiunto \n per stampare su nuove righe. Notare come il valore predefinito non viene stampato. È solo quando si fa clic sul controllo che viene chiamata la funzione di callback. Facendo clic sulla freccia giù con un valore predefinito di 0, possiamo stampare il print0 valore.

Infine, limitiamo i valori disponibili a un set hardcoded. Questo potrebbe anche essere letto da un'origine dati (ad esempio, un file di testo o XML).

Rilievo, aspetto incavato e rialzato dei widget

Possiamo controllare l'aspetto del nostro Spinbox widget utilizzando una proprietà che li fa apparire in diversi formati infossati o rialzati.

Prepararsi

Ne aggiungeremo un altro Spinbox controllo per dimostrare gli aspetti disponibili dei widget, utilizzando la proprietà rilievo del Spinbox controllo.

Come farlo...

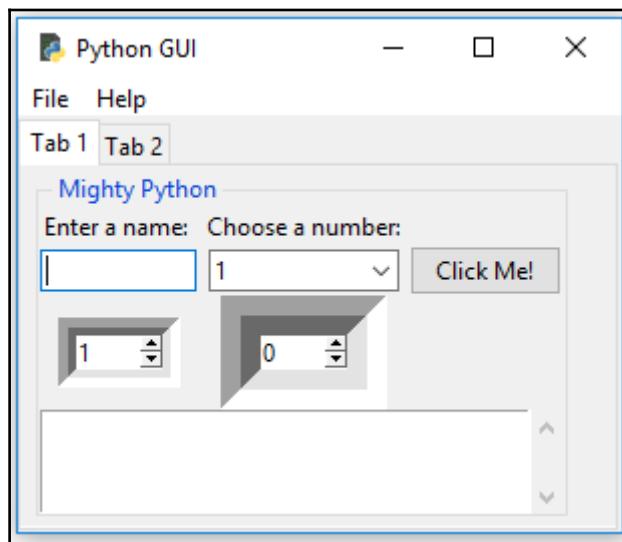
Per prima cosa, aumentiamo il larghezza del bordo per distinguere il nostro secondo Spinbox dal primo

Casella rotante:

```
# Aggiunta di un secondo widget Spinbox
spin = Spinbox(mighty, values=(0, 50, 100), width=5, bd=20,
                comando=_spin)
spin.grid(columna=1, riga=2)
```

Questo creerà il seguente output della GUI:

GUI_spinbox_two_sunken.py



Entrambi i nostri precedenti Spinbox i widget hanno lo stesso stile di rilievo. L'unica differenza è che il nostro nuovo widget a destra del primoSpinbox ha una larghezza del bordo molto maggiore.

Nel nostro codice, non abbiamo specificato quale proprietà di rilievo usare, quindi lo scarico è di default tk.SUNKEN.



Abbiamo importato tkinter come tk. Questo è il motivo per cui possiamo chiamare la proprietà di rilievo come tk.SUNKEN.

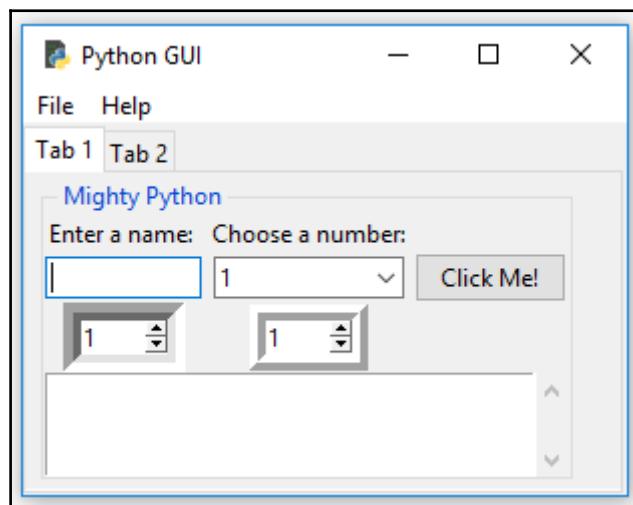
Di seguito sono riportate le opzioni delle proprietà di rilievo disponibili che possono essere impostate:

tk.SUNKEN	tk.RAISED	tk.FLAT	tk.GROOVE	tk.RIDGE	
-----------	-----------	---------	-----------	----------	--

Assegnando le diverse opzioni disponibili alla proprietà rilievo, possiamo creare diversi aspetti per questo widget.

Assegnando il tk.RIDGE rilievo e riducendo la larghezza del bordo allo stesso valore del nostro primo Spinbox widget risulta nella seguente GUI:

GUI_spinbox_two_ridge.py



Come funziona...

Primo, abbiamo creato un secondo Spinbox allineato nella seconda colonna (indice == 1). Il valore predefinito è affondato, quindi sembra simile al nostro primo Spinbox. Abbiamo distinto i due widget aumentando la larghezza del bordo del secondo controllo (quello a destra).

Successivamente, impostiamo esplicitamente la proprietà rilievo di the Spinbox aggeggio. Abbiamo reso la larghezza del bordo uguale alla nostra prima Spinbox perché, dandogli un rilievo diverso, le differenze diventavano visibili senza dover modificare altre proprietà.

Ecco un esempio delle diverse opzioni:

```
# Adding a second Spinbox widget displaying its relief options with larger borderline
# uncomment each next code line to see the different effects
# spin = Spinbox(mighty, values=(0, 50, 100), width=5, bd=20, command=_spin)    # default value is: tk.SUNKEN
# spin = Spinbox(mighty, values=(0, 50, 100), width=5, bd=20, command=_spin, relief=tk.FLAT)
# spin = Spinbox(mighty, values=(0, 50, 100), width=5, bd=20, command=_spin, relief=tk.RAISED)
# spin = Spinbox(mighty, values=(0, 50, 100), width=5, bd=20, command=_spin, relief=tk.SUNKEN) # default
# spin = Spinbox(mighty, values=(0, 50, 100), width=5, bd=20, command=_spin, relief=tk.GROOVE)
# spin = Spinbox(mighty, values=(0, 50, 100), width=5, bd=20, command=_spin, relief=tk.RIDGE)
```

Creazione di tooltip usando Python

Questa ricetta ci mostrerà come creare tooltip. Quando l'utente passa il mouse su un widget, saranno disponibili ulteriori informazioni sotto forma di suggerimento.

Codificheremo queste informazioni aggiuntive nella nostra GUI.

Prepararsi

Aggiungeremo funzionalità più utili alla nostra GUI. Sorprendentemente, aggiungere un tooltip ai nostri controlli dovrebbe essere semplice, ma non è così semplice come vorremmo che fosse.

Per ottenere questa funzionalità desiderata, inseriremo il nostro codice del tooltip nella propria classe OOP.

Come farlo...

Aggiungi la seguente classe appena sotto le istruzioni di importazione:

```
#=====
class ToolTip(object):
    def __init__(self, widget):
        self.widget = widget
        self.tip_window = None

    def show_tip(self, tip_text):
        "Display text in a tooltip window"
        if self.tip_window or not tip_text:
            return
        x, y, _cx, _cy = self.widget.bbox("insert")      # get size of widget
        x = x + self.widget.winfo_rootx() + 25          # calculate to display tooltip
        y = y + _cy + self.widget.winfo_rooty() + 25     # below and to the right
        self.tip_window = tw = tk.Toplevel(self.widget)  # create new tooltip window
        tw.wm_overrideredirect(True)                     # remove all Window Manager (wm) decorations
#        tw.wm_overrideredirect(False)                   # uncomment to see the effect
        tw.wm_geometry("+%d+%d" % (x, y))              # create window size

        label = tk.Label(tw, text=tip_text, justify=tk.LEFT,
                         background="#ffffe0", relief=tk.SOLID, borderwidth=1,
                         font=("tahoma", "8", "normal"))
        label.pack(ipadx=1)

    def hide_tip(self):
        tw = self.tip_window
        self.tip_window = None
        if tw:
            tw.destroy()

#=====
def create_ToolTip(widget, text):
    toolTip = ToolTip(widget)           # create instance of class
    def enter(event):
        toolTip.show_tip(text)
    def leave(event):
        toolTip.hide_tip()
    widget.bind('<Enter>', enter)    # bind mouse events
    widget.bind('<Leave>', leave)
```

In un **Programmazione orientata agli oggetti (OOP)** approccio, creiamo una nuova classe nel nostro modulo Python. Python ci consente di inserire più di una classe nello stesso modulo Python e ci consente anche di *mescolare e abbinare* classi e funzioni regolari nello stesso modulo.

Il codice precedente fa esattamente questo. Il comando `class` è una classe Python e per usarla dobbiamo istanziarla.

Se non hai familiarità con la programmazione OOP, creare un'istanza di un oggetto per creare un'istanza della classe può sembrare piuttosto noioso.

Il principio è abbastanza semplice e molto simile alla creazione di una funzione Python tramite a def istruzione e poi, più avanti nel codice, chiamando effettivamente questa funzione.

In modo molto simile, creiamo prima un progetto di una classe e lo assegniamo semplicemente a una variabile aggiungendo parentesi al nome della classe, come segue:

```
classe AClass():
    passaggio
    istanza_di_una_classe = AClass()
    print(istanza_di_a_classe)
```

Il codice precedente stampa un indirizzo di memoria e mostra anche che la nostra variabile ora ha un riferimento a questa istanza di classe.

La cosa interessante di OOP è che possiamo creare molte istanze della stessa classe.

Nel nostro codice del tooltip, dichiariamo una classe Python e la facciamo esplicitamente ereditare da oggetto, che è il fondamento di tutte le classi Python. Possiamo anche tralasciarlo, come abbiamo fatto nel Una classe esempio di codice, perché è l'impostazione predefinita per tutte le classi Python.

Dopo tutto il necessario codice di creazione del tooltip che si verifica all'interno del Descrizione comando class, possiamo alla programmazione Python non OOP creando una funzione appena sotto di essa.

Definiamo il create_ToolTip() funzione e si aspetta che uno dei nostri widget GUI venga passato come argomento, in modo che possiamo visualizzare un suggerimento quando passiamo il mouse su questo controllo.

Il create_ToolTip() la funzione crea effettivamente una nuova istanza del nostro Descrizione comando class per ogni widget per cui lo chiamiamo.

Possiamo aggiungere un tooltip per il nostro Spinbox widget, come segue:

```
# Aggiungi un suggerimento Tool
create_ToolTip(spin, 'Questo è un controllo Spin')
```

Potremmo fare lo stesso per tutti gli altri nostri widget GUI nello stesso modo. Non ci resta che passare un riferimento al widget che desideriamo avere un tooltip, visualizzando alcune informazioni extra. Per noi Testo scorrevole widget, abbiamo creato il scorrere variabile puntano ad esso, quindi questo è ciò che passiamo al costruttore della nostra funzione di creazione del tooltip:

```
# Using a scrolled Text control
scrol_w = 30
scrol_h = 3
scrol = scrolledtext.ScrolledText(mighty, width=scrol_w, height=scrol_h, wrap=tk.WORD)
scrol.grid(column=0, row=3, sticky='WE', columnspan=3)

# Add a Tooltip to the ScrolledText widget
create_ToolTip(scrol, 'This is a ScrolledText widget')
```

Come funziona...

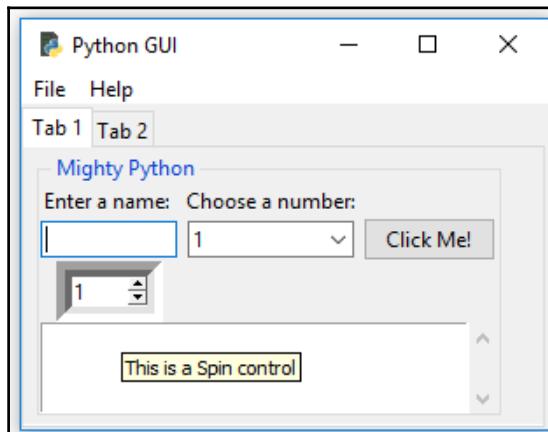
Questo è l'inizio della programmazione OOP in questo libro. Questo potrebbe sembrare un po' avanzato, ma non preoccuparti; spiegheremo tutto e funziona davvero.

Considera l'aggiunta del seguente codice appena sotto la creazione del Casella rotante:

```
# Aggiungi un suggerimento Tool
create_ToolTip(spin, 'Questo è un controllo Spin.')
```

Ora, quando passiamo il mouse sul widget Spinbox, otteniamo un tooltip, che fornisce informazioni aggiuntive all'utente:

GUI_tooltip.py



Chiamiamo la funzione che crea il Descrizione comando, e poi passiamo un riferimento al widget e il testo che vogliamo visualizzare quando passiamo il mouse sopra il widget.

Il resto delle ricette in questo libro utilizzerà OOP quando ha senso. Qui abbiamo mostrato l'esempio OOP più semplice possibile. Come impostazione predefinita, ogni classe Python che creiamo eredita daloggetto classe base. Python, essendo il linguaggio di programmazione pragmatico che è veramente, semplifica il processo di creazione della classe.

Possiamo scrivere la seguente sintassi:

classe ToolTip (oggetto):

 passaggio

Possiamo anche semplificarlo lasciando fuori la classe base predefinita:

classe ToolTip():

 passaggio

Allo stesso modo, possiamo ereditare ed espandere qualsiasi tkinter classe.

Aggiunta di una barra di avanzamento alla GUI

In questa ricetta, aggiungeremo un Barra di avanzamento alla nostra GUI. È molto facile aggiungere un ttk.Progressbar, e dimostreremo come avviare e fermare un Barra di avanzamento. Questa ricetta ti mostrerà anche come ritardare l'arresto di una Barra di avanzamento e come eseguirlo in un ciclo.



Barra di avanzamento viene in genere utilizzato per mostrare lo stato corrente di un processo di lunga durata.

Prepararsi

Aggiungeremo la barra di avanzamento a **Scheda 2** della GUI che abbiamo sviluppato nella ricetta precedente: *Creazione di tooltip usando Python*.

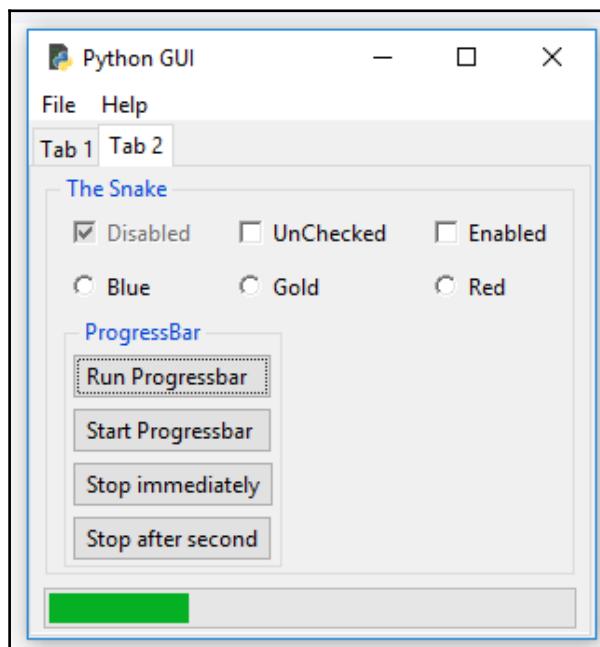
Come farlo...

Innanzitutto, aggiungiamo quattro pulsanti in EtichettaFrame sopra **Scheda 2**, sostituendo le etichette che c'erano prima. Impostiamo il Cornice per etichette proprietà del testo a Barra di avanzamento.

Quindi posizioniamo a ttk.Progressbar widget sotto tutti gli altri widget su **Scheda 2** e allinea questo nuovo widget con gli altri widget.

La nostra GUI ora ha il seguente aspetto:

GUI_progressbar.py



Colleghiamo ciascuno dei nostri quattro nuovi pulsanti a una nuova funzione di callback, che assegniamo alla loro proprietà di comando:

```
# Add Buttons for Progressbar commands
ttk.Button(buttons_frame, text=" Run Progressbar ", command=run_progressbar).grid(column=0, row=0, sticky='W')
ttk.Button(buttons_frame, text=" Start Progressbar ", command=start_progressbar).grid(column=0, row=1, sticky='W')
ttk.Button(buttons_frame, text=" Stop immediately ", command=stop_progressbar).grid(column=0, row=2, sticky='W')
ttk.Button(buttons_frame, text=" Stop after second ", command=progressbar_stop_after).grid(column=0, row=3, sticky='W')
```



Facendo clic su **Esegui barra di avanzamento** il pulsante eseguirà il Barra di avanzamento da sinistra a destra, poi il Barra di avanzamento si fermerà lì e la barra verde scomparirà.

Ecco il codice:

```
# Add a Progressbar to Tab 2
progress_bar = ttk.Progressbar(tab2, orient='horizontal', length=286, mode='determinate')
progress_bar.grid(column=0, row=3, pady=2)

# update progressbar in callback loop
def run_progressbar():
    progress_bar["maximum"] = 100
    for i in range(101):
        sleep(0.05)
        progress_bar["value"] = i    # increment progressbar
        progress_bar.update()      # have to call update() in loop
    progress_bar["value"] = 0      # reset/clear progressbar
```

Facendo clic su **Avvia barra di avanzamento** il pulsante avvierà il Barra di avanzamento. Il Barra di avanzamento durerà fino alla fine, e poi ricomincerà tutto da sinistra, in un ciclo infinito:

```
def start_progressbar():
    progress_bar.start()
```

Per fermare questo ciclo infinito del nostro widget della barra di avanzamento, creiamo semplicemente un'altra funzione di callback e la assegniamo a uno dei nostri pulsanti:

```
def stop_progressbar():
    progress_bar.stop()
```

Non appena clicchiamo su **Interrompi barra di avanzamento** pulsante, il nostro Barra di avanzamento si fermerà e si ripristinerà dall'inizio, rendendolo invisibile. Non vediamo più una barra verde all'interno il Barra di avanzamento.



Se clicchiamo su **Esegui barra di avanzamento** pulsante e quindi fare clic sul **Interrompi barra di avanzamento**, l'avanzamento della barra si interromperà temporaneamente, ma poi il ciclo verrà eseguito fino al completamento e così anche la ProgressBar.

Possiamo anche ritardare l'arresto della corsa Barra di avanzamento. Mentre potremmo aspettarci che a dormire affermazione farebbe il trucco, non lo fa. Invece, dobbiamo usare la funzione `dopo` di tkinter, come segue:

```
def progressbar_stop_after(wait_ms=1000):
    win.after(wait_ms, progress_bar.stop)
```

Codificarlo in questo modo fermerà la corsa Barra di avanzamento quando si fa clic su questo pulsante dopo il tempo specificato nel `wait_ms` variabile, in millisecondi.

Come funziona...

Possiamo specificare un valore massimo e utilizzare questo valore in un ciclo, insieme a una dichiarazione.

Possiamo avviare e fermare il progresso del Barra di avanzamento, usando il inizio e fermare comandi integrati nel Barra di avanzamento aggiungendo.

Possiamo anche ritardare l'arresto del progresso nella barra usando il built-in di tkinter `dopo` funzione.

Lo facciamo chiamando il `dopo` funzione sul riferimento alla nostra finestra principale della GUI, che abbiamo chiamato `vincere`.

Come usare il widget della tela

Questa ricetta mostra come aggiungere effetti cromatici drammatici alla nostra GUI utilizzando il widget canvas di tkinter.

Prepararsi

Miglioreremo il nostro codice precedente da `GUI_tooltip.py`, e miglioreremo l'aspetto della nostra GUI aggiungendo altri colori.

Come farlo...

Innanzitutto, creeremo una terza scheda nella nostra GUI per isolare il nostro nuovo codice.

Ecco il codice per creare la nuova terza scheda:

```
tabControl = ttk.Notebook(win) # Crea controllo scheda  
  
tab1 = ttk.Frame(tabControl) # Crea una scheda  
tabControl.add(tab1, text='Tab 1') # Aggiungi la scheda  
  
tab2 = ttk.Frame(tabControl) # Aggiungi una seconda scheda  
tabControl.add(tab2, text='Tab 2')  
  
tab3 = ttk.Frame(tabControl) # Aggiungi una terza scheda  
tabControl.add(tab3, text='Tab 3')  
  
tabControl.pack(expand=1, fill="both") # Pack per rendere visibili le schede
```

Successivamente, usiamo un altro widget integrato di tkinter: tela. A molte persone piace questo widget perché ha potenti capacità:

```
# Controllo scheda 3 -----  
tab3_frame = tk.Frame(tab3, bg='blu')  
tab3_frame.pack()  
per orange_color nell'intervallo (2):  
    canvas = tk.Canvas(tab3_frame, larghezza=150, altezza=80,  
                      highlightthickness=0, bg='arancione')  
    canvas.grid(riga=colore_arancio, colonna=colore_arancio)
```

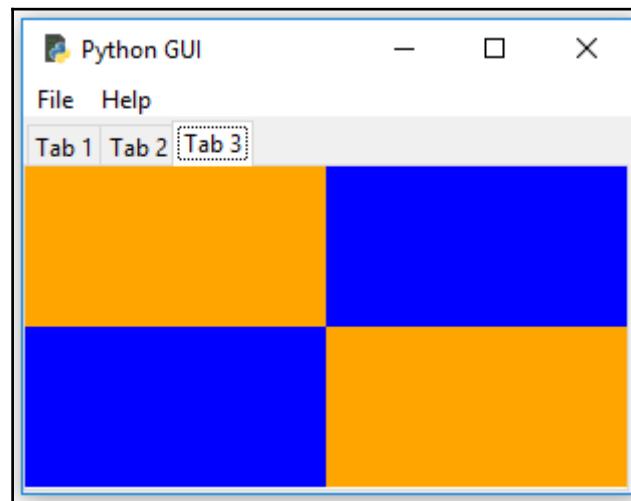
Come funziona...

Dopo aver creato la nuova scheda, posizioniamo un normale tk.Frame in esso e assegnargli un colore di sfondo blu. Nel ciclo, creiamo due tk.Canvas widget, rendendo il loro colore arancione e assegnandoli alle coordinate della griglia 0,0 e 1,1. Questo rende anche il colore di sfondo blu del tk.Frame visibile nelle altre due posizioni della griglia.

Lo screenshot seguente mostra il risultato creato eseguendo il codice precedente e facendo clic sul nuovo **Scheda 3**. È davvero arancione e blu quando si esegue il codice. In un libro stampato non a colori, questo potrebbe non essere visivamente ovvio, ma quei colori sono veri; puoi fidarti di me su questo.

Puoi verificare le funzionalità di grafica e disegno effettuando una ricerca online. Non approfondirò questo widget in questo libro (ma è molto interessante):

GUI_canvas.py



4

Dati e classi

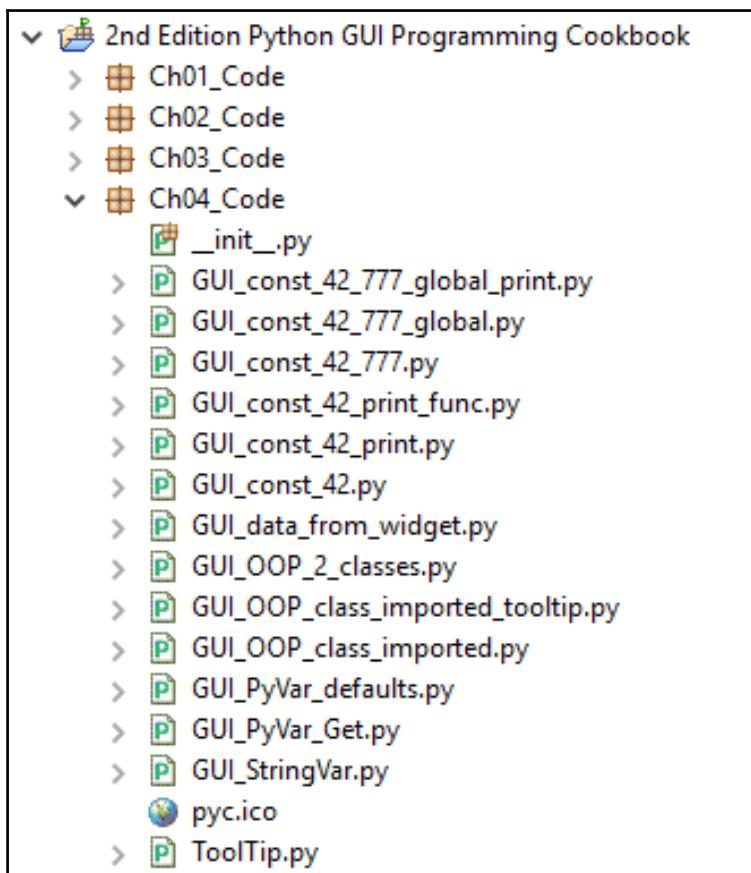
In questo capitolo, useremo i dati e le classi OOP usando Python 3.6 e versioni successive.
Tratteremo le seguenti ricette:

- Come usare StringVar()
- Come ottenere dati da un widget utilizzando
- variabili globali a livello di modulo
- In che modo la codifica nelle classi può migliorare le funzioni
- di callback di scrittura della GUI
- Creazione di componenti GUI riutilizzabili

introduzione

In questo capitolo, salveremo i dati della nostra GUI in tkinter variabili. Inizieremo anche a utilizzare l'OOP per estendere l'esistente tkinter classi per estendere le funzionalità integrate di tkinter. Questo ci porterà a creare componenti OOP riutilizzabili.

Ecco la panoramica dei moduli Python per questo capitolo:



Come usare StringVar()

Ci sono tipi di programmazione incorporati in tkinter che differiscono leggermente dai tipi Python con cui siamo abituati a programmare. StringVar() è uno di questi tipi di tkinter. Questa ricetta ti mostrerà come usare ilStringVar() genere.

Prepararsi

Imparerai come salvare i dati dalla GUI di tkinter in variabili in modo che possiamo usare quei dati. Possiamo impostare e ottenere i loro valori, che è molto simile ai metodi getter/setter di Java.

Ecco alcuni dei tipi di codifica disponibili in tkinter:

strVar = StringVar()	# Contiene una stringa; il valore predefinito è una stringa vuota ""
intVar = IntVar()	# Contiene un numero intero; il valore predefinito è 0
dbVar = DoubleVar()	# Contiene un galleggiante; il valore predefinito è 0.0
blVar = BooleanVar()	# Contiene un valore booleano, restituisce 0 per False e 1 per True



Lingue diverse chiamano numeri con punti decimali come galleggia o raddoppia. Tkinter li chiama DoubleVar, ciò che è noto in Python come galleggiante tipo di dati. A seconda del livello di precisione, i dati float e double possono essere diversi. Ecco, stiamo traducendo tkinterDoubleVar in ciò che Python si trasforma in un Python galleggiante genere.

Questo diventa più chiaro quando aggiungiamo a DoubleVar con un float Python e guarda il tipo risultante, che è un Python galleggiante e non più a Doppia Var:

GUI_PyDoubleVar_to_Float_Get.py

The screenshot shows the Eclipse IDE interface with a code editor and a terminal window. The code editor contains the following Python script:

```
import tkinter as tk

# Create instance of tkinter
win = tk.Tk()

# Create DoubleVar
doubleData = tk.DoubleVar()
print(doubleData.get())          # default value
doubleData.set(2.4)
print(type(doubleData))

add_doubles = 1.2222222222222222 + doubleData.get()
print(add_doubles)
print(type(add_doubles))
```

The terminal window (Console) displays the following output:

```
<terminated> C:\Eclipse_NEON_workspace\2nd Edition Python GUI Programming Cookbook
0.0
<class 'tkinter.DoubleVar'>
3.622222222222222
<class 'float'>
```

Come farlo...

Creeremo un nuovo modulo Python e lo screenshot seguente mostra sia il codice che l'output risultante:

GUI_StringVar.py

```
import tkinter as tk

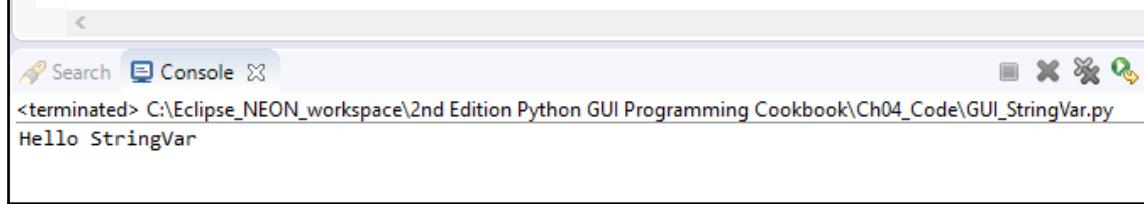
# Create instance of tkinter
win = tk.Tk()

# Assign tkinter Variable to strData variable
strData = tk.StringVar()

# Set strData variable
strData.set('Hello StringVar')

# Get value of strData variable
varData = strData.get()

# Print out current value of strData
print(varData)
```



The screenshot shows the Eclipse IDE's Console view. The title bar says "Console". The content area shows the command "<terminated> C:\Eclipse_NEON_workspace\2nd Edition Python GUI Programming Cookbook\Ch04_Code\GUI_StringVar.py" and the output "Hello StringVar".

Per prima cosa, importiamo il tkinter modulo e alias al nome tk.

Successivamente, usiamo questo alias per creare un'istanza di Tk classe aggiungendo parentesi a Tk, che chiama il costruttore della classe. Questo è lo stesso meccanismo della chiamata di una funzione; solo qui, creiamo un'istanza di una classe.

Di solito, usiamo questa istanza assegnata a vincere variabile per avviare il ciclo dell'evento principale più avanti nel codice, ma qui non stiamo visualizzando una GUI, piuttosto, stiamo dimostrando come usare il tkinter StringVar genere.



Dobbiamo ancora creare un'istanza di Tk(). Se commentiamo questa riga, otterremo un errore da tkinter, quindi questa chiamata è necessaria.

Quindi, creiamo un'istanza di tkinter StringVar digita e assegnalo al nostro Python strData variabile.

Dopodiché, usiamo la nostra variabile per chiamare il impostato() metodo attivo StringVar e dopo aver impostato un valore, otteniamo il valore, lo salviamo in una nuova variabile denominata varData, e poi stamparne il valore.

Nella console Eclipse PyDev, verso la parte inferiore dello screenshot, possiamo vedere l'output stampato sulla console, che è **Ciao StringVar**.

Successivamente, stamperemo i valori predefiniti di tkinter IntVar, DoubleVar, e Variabile booleana tipi:

GUI_PyVar_defaults.py

The screenshot shows the Eclipse PyDev interface with the 'Console' tab selected. The console window displays the following text:

```
# Get value of strData variable
varaData = strData.get()

# Print out current value of strData
print(varaData)

# Print out the default tkinter variable values
print(tk.IntVar())
print(tk.DoubleVar())
print(tk.BooleanVar())
<
```

<terminated> C:\Eclipse_NEON_workspace\2nd Edition Python GUI Programming Cookbook\Ch04_Code\GUI_PyVar_defaults.py

Hello StringVar
PY_VAR1
PY_VAR2
PY_VAR3

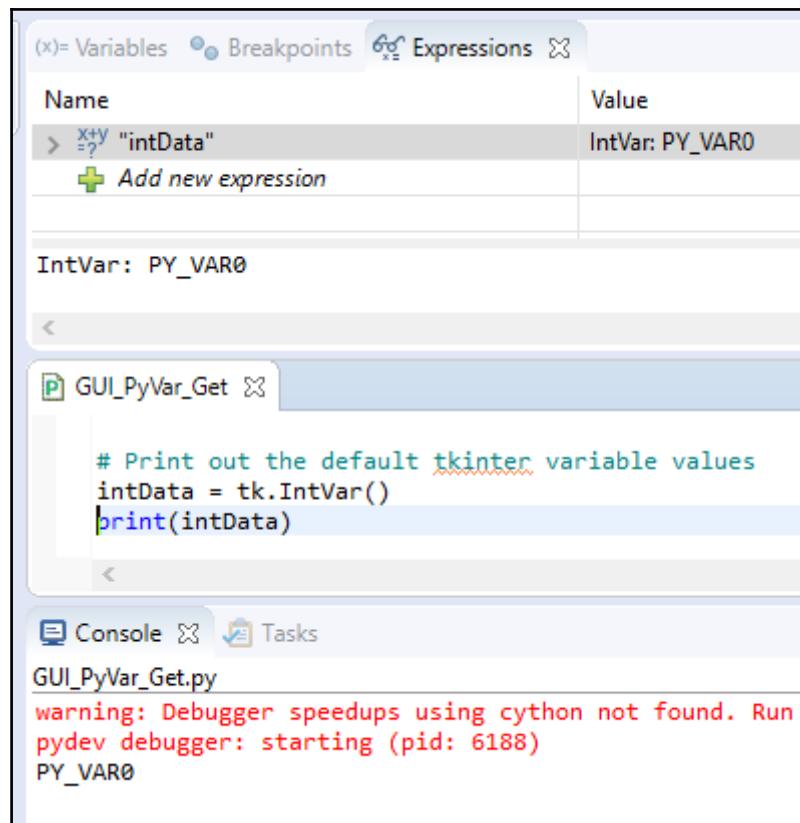
Come funziona...

Come si può vedere nello screenshot precedente, i valori di default non vengono stampati, come ci saremmo aspettati.

La letteratura online menziona i valori predefiniti ma non vedremo quei valori finché non chiameremo il otteneri metodo su di essi. Altrimenti, otteniamo solo un nome di variabile che incrementa automaticamente (ad esempio, PY_VAR3, come si può vedere nello screenshot precedente).

Assegnando il tkinter type in una variabile Python non cambia il risultato. Non otteniamo ancora il valore predefinito.

Qui, ci stiamo concentrando sul codice più semplice (che crea PY_VAR0):



Il valore è PY_VAR0, non il previsto 0, finché non chiamiamo il ottenere metodo. Ora possiamo vedere il valore predefinito. Non abbiamo chiamato impostato, quindi vediamo il valore predefinito assegnato automaticamente a ciascun tipo di tkinter una volta che chiamiamo il ottenere metodo su ogni tipo:

GUI_PyVar_Get.py

The screenshot shows the Eclipse PyDev debugger interface. At the top, there's a toolbar with tabs for Variables, Breakpoints, Expressions, and others. Below the toolbar is a table showing variable values:

Name	Value
> <code>intData</code>	IntVar: PY_VAR0
> <code>intData.get()</code>	int: 0
<code>Add new expression</code>	

Below the table, the variable `int` is shown with a value of 0. The bottom part of the window is the Python code being run:

```
# Print out the default tkinter variable values
intData = tk.IntVar()
print(intData)
print(intData.get())
```

At the bottom, the Console tab is active, showing the output of the script:

```
warning: Debugger speedups using cython not found. Run pydev debugger: starting (pid: 7984)
PY_VAR0
0
```

Nota come il valore predefinito di 0 viene stampato sulla console per il IntVar esempio che abbiamo salvato nel in intData variabile. Possiamo anche vedere i valori nella finestra del debugger Eclipse PyDev nella parte superiore dello screenshot.

Come ottenere dati da un widget

Quando l'utente inserisce i dati, vogliamo fare qualcosa con essi nel nostro codice. Questa ricetta mostra come acquisire dati in una variabile. Nella ricetta precedente ne abbiamo create diverse `tkinter` variabili di classe. Erano autonomi. Ora li stiamo collegando alla nostra GUI, utilizzando i dati che otteniamo dalla GUI e archiviandoli nelle variabili Python.

Prepararsi

Continueremo a utilizzare la GUI Python in cui stavamo costruendo Capitolo 3, *Look and Feel personalizzazione*. Riutilizzeremo e migliorneremo il codice di `GUI_progressbar.py` da quel capitolo.

Come farlo...

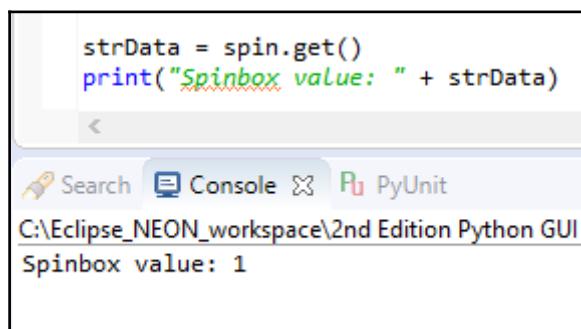
Assegneremo un valore dalla nostra GUI a una variabile Python.

Aggiungi il seguente codice nella parte inferiore del nostro modulo, appena sopra il ciclo dell'evento principale:

```
strData = spin.get()
print("Valore casella numerica: " + strData)

# Posiziona il cursore nella voce del nome
nome_inserito.focus()
# =====
# Avvia GUI
# =====
win.mainloop()
```

L'esecuzione del codice ci dà il seguente risultato:



```
strData = spin.get()
print("Spinbox value: " + strData)

< 
Search Console PyUnit
C:\Eclipse_NEON_workspace\2nd Edition Python GUI
Spinbox value: 1
```

Recupereremo il valore corrente di Spinbox controllo.



Abbiamo posizionato il nostro codice sopra il ciclo dell'evento principale della GUI, quindi la stampa avviene prima che la GUI diventi visibile. Dovremmo inserire il codice in una funzione di callback se volessimo stampare il valore corrente dopo aver visualizzato la GUI e aver modificato il valore del Spinbox controllo.

Abbiamo creato il nostro Spinbox widget utilizzando il seguente codice, codificando i valori disponibili in esso:

```
# Aggiunta di un widget Spinbox utilizzando un insieme di valori
spin = Spinbox(potente, valori = (1, 2, 4, 42, 100), larghezza = 5, bd = 8,
                comando=_spin)
spin.grid(colonna=0, riga=2)
```

Possiamo anche spostare l'hardcoding dei dati fuori dalla creazione del Spinbox istanza di classe e impostala in seguito:

```
# Aggiunta di un widget Spinbox che assegna valori dopo la creazione
spin = Spinbox(mighty, width=5, bd=8, command=_spin) spin['values']
= (1, 2, 4, 42, 100)
spin.grid(colonna=0, riga=2)
```

Non importa come creiamo il nostro widget e inseriamo i dati in esso perché possiamo accedere a questi dati utilizzando il ottener() metodo sull'istanza del widget.

Come funziona...

Per ottenere i valori dalla nostra GUI scritta usando tkinter, usiamo il tkinter ottener() metodo su un'istanza del widget da cui desideriamo ottenere il valore.

Nell'esempio precedente, abbiamo usato il Spinbox controllo, ma il principio è lo stesso per tutti i widget che hanno a ottener() metodo.

Una volta ottenuti i dati, siamo in un mondo di puro Python e tkinter ci ha servito bene nella costruzione della nostra GUI. Ora che sappiamo come estrarre i dati dalla nostra GUI, possiamo usare questi dati.

Utilizzo di variabili globali a livello di modulo

L'incapsulamento è un punto di forza in qualsiasi linguaggio di programmazione, poiché ci consente di programmare utilizzando l'OOP. Python è sia OOP che procedurale. Possiamo creare globale variabili che sono localizzate nel modulo in cui risiedono. Sono globale solo a questo modulo, che è una forma di incapsulamento. Perché vogliamo questo? Perché man mano che aggiungiamo sempre più funzionalità alla nostra GUI, vogliamo evitare conflitti di denominazione che potrebbero causare bug nel nostro codice.



Non vogliamo nominare scontri che creano bug nel nostro codice! Gli spazi dei nomi sono un modo per evitare questi bug e, in Python, possiamo farlo utilizzando i moduli Python (che sono spazi dei nomi non ufficiali).

Prepararsi

Possiamo dichiarare a livello di modulo **globali** in qualsiasi modulo appena sopra e al di fuori delle funzioni.

Dobbiamo quindi usare il globale Parola chiave Python per fare riferimento a loro. Se ci dimentichiamo di usare globale nelle funzioni, creeremo accidentalmente nuove variabili locali. Questo sarebbe un bug e qualcosa che davvero non vogliamo fare.



Python è un linguaggio dinamico e fortemente tipizzato. Noteremo bug come questo (dimenticando di definire l'ambito delle variabili con il globale parola chiave) solo in fase di esecuzione.

Come farlo...

Aggiungi il codice mostrato alla riga 17 alla GUI che abbiamo usato nella ricetta precedente, *Come ottenere dati da un widget*, creando una variabile globale a livello di modulo. Usiamo la convenzione in maiuscolo in stile C, che non è veramente Pythonic, ma penso che questo enfatizzi il principio che stiamo affrontando in questa ricetta:

```
6#=====
7 # imports
8 =====
9import tkinter as tk
10from tkinter import ttk
11from tkinter import scrolledtext
12from tkinter import Menu
13from tkinter import messagebox as msg
14from tkinter import Spinbox
15from time import sleep      # careful - this can freeze the GUI
16
17GLOBAL_CONST = 42
18
```

L'esecuzione del codice comporta una stampa del globale. Nota **42** in corso di stampa sulla console Eclipse:

GUI_const_42_print.py

The screenshot shows the Eclipse IDE interface. In the top-left, there's a code editor window containing Python code. In the bottom-right, there's a 'Console' tab in the Eclipse interface, which displays the output of the program execution. The output shows the value '42' printed to the console.

```
213 # Printing the Global works
214 print(GLOBAL_CONST)
215
216 name_entered.focus()
217=====
218 # Start GUI
219 =====
220 win.mainloop()

<!--
  Search  Console  PyUnit
  <terminated> C:\Eclipse_NEON_workspace\2nd Edition
  42
-->
```

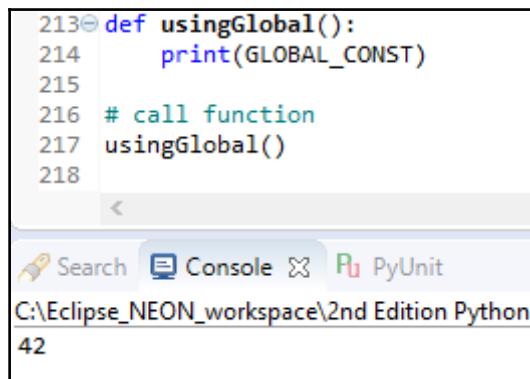
Come funziona...

Definiamo a globale variabile nella parte superiore del nostro modulo e stampiamo il suo valore in seguito, verso la parte inferiore del nostro modulo.

Che funziona.

Aggiungi questa funzione verso la parte inferiore del nostro modulo:

GUI_const_42_print_func.py

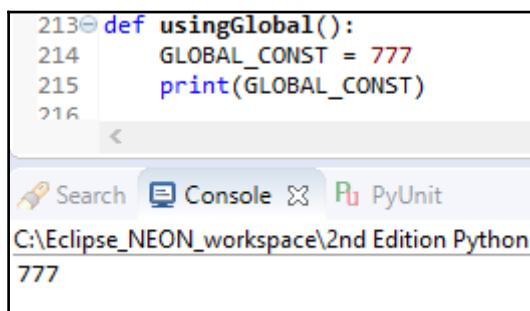


```
213 def usingGlobal():
214     print(GLOBAL_CONST)
215
216 # call function
217 usingGlobal()
218
```

The screenshot shows the Eclipse IDE interface with the Python perspective. A code editor window displays the above Python code. Below it, the 'Console' view is active, showing the output of the code execution. The output shows the value '42' printed to the console.

Nel frammento di codice precedente, usiamo il livello di modulo globale. È facile commettere un errore con l'ombra globale, come dimostrato di seguito:

GUI_const_42_777.py



```
213 def usingGlobal():
214     GLOBAL_CONST = 777
215     print(GLOBAL_CONST)
216
```

The screenshot shows the Eclipse IDE interface with the Python perspective. A code editor window displays the above Python code. Below it, the 'Console' view is active, showing the output of the code execution. The output shows the value '777' printed to the console, illustrating that the local variable 'GLOBAL_CONST' shadows the global variable.

Nota come **42** diventa **777**, anche se stiamo usando lo stesso nome di variabile.



Non esiste un compilatore in Python che ci avvisi se sovrascriviamo globale variabili in una funzione locale. Ciò può causare difficoltà nel debug in fase di esecuzione.

Senza usare il globale qualificatore (riga 214), otteniamo un errore.

The screenshot shows the Eclipse IDE interface. On the left is a code editor window containing the following Python code:

```
213 def usingGlobal():
214     #     global GLOBAL_CONST
215     print(GLOBAL_CONST)
216     GLOBAL_CONST = 777
217     print(GLOBAL_CONST)
218
219
220 # call function
221 usingGlobal()
222
```

On the right is a terminal window titled "Console" showing the execution results and an error message:

```
<terminated> C:\Eclipse_NEON_workspace\2nd Edition Python GUI Programming Cookbook\Ch04_Code\
Traceback (most recent call last):
File "C:\Eclipse_NEON_workspace\2nd Edition Python GUI Programming Cookbook\Ch
    usingGlobal()
File "C:\Eclipse_NEON_workspace\2nd Edition Python GUI Programming Cookbook\Ch
    print(GLOBAL_CONST)
UnboundLocalError: local variable 'GLOBAL_CONST' referenced before assignment
```

Quando qualifichiamo la nostra variabile locale con il globale parola chiave, possiamo stampare il valore di globale variabile oltre a sovrascrivere questo valore localmente:

The screenshot shows the Eclipse IDE interface. On the left is a code editor window containing the same Python code as before, but with the addition of the "global" keyword in line 214:

```
213 def usingGlobal():
214     global GLOBAL_CONST
215     print(GLOBAL_CONST)
216     GLOBAL_CONST = 777
217     print(GLOBAL_CONST)
218
219
220 # call function
221 usingGlobal()
222
```

On the right is a terminal window titled "Console" showing the execution results:

```
C:\Eclipse_NEON_workspace\2nd Edition Python\
42
777
```

Il globale le variabili possono essere molto utili quando si programmano piccole applicazioni. Possono aiutarci a rendere disponibili i dati attraverso metodi e funzioni all'interno dello stesso modulo Python e talvolta il sovraccarico dell'OOP non è giustificato.

Man mano che i nostri programmi crescono in complessità, i vantaggi che otteniamo dall'utilizzo dei globali possono diminuire rapidamente.



È meglio evitare le variabili globali e l'ombreggiatura accidentale utilizzando lo stesso nome in ambiti diversi. Possiamo usare OOP invece di usare variabili globali.

Abbiamo giocato con il globale variabili all'interno del codice procedurale e imparato come può portare a bug difficili da eseguire il debug. Nella prossima ricetta, passeremo all'OOP, che può eliminare tali bug.

In che modo la codifica nelle classi può migliorare la GUI

Finora, abbiamo codificato in uno stile procedurale. Questo è un metodo di scripting rapido di Python. Una volta che il nostro codice diventa sempre più grande, dobbiamo passare alla codifica in OOP.

Perché?

Perché, tra molti altri vantaggi, l'OOP ci consente di spostare il codice utilizzando i metodi. Una volta che usiamo le classi, non dobbiamo più posizionare fisicamente il codice sopra il codice che lo chiama. Questo ci dà una grande flessibilità nell'organizzazione del nostro codice.

Possiamo scrivere codice correlato accanto ad altro codice e non dobbiamo più preoccuparci che il codice non venga eseguito perché il codice non si trova sopra il codice che lo chiama.

Possiamo portarlo ad alcuni estremi piuttosto fantasiosi codificando moduli che fanno riferimento a metodi che non vengono creati all'interno di quel modulo. Si basano sullo stato di runtime che ha creato quei metodi durante l'esecuzione del codice.



Se i metodi che chiamiamo non sono stati creati in quel momento, otteniamo un errore di runtime.

Prepararsi

Trasformeremo il nostro intero codice procedurale in OOP in modo molto semplice. Lo trasformiamo semplicemente in una classe, indentiamo tutto il codice esistente e lo anteponiamose stesso a tutte le variabili.

È molto facile.

Anche se all'inizio potrebbe sembrare un po' fastidioso dover anteporre tutto al `with` se stesso parola chiave che rende il nostro codice più prolioso (ehi, stiamo sprecando così tanta carta...), alla fine ne vale la pena.

Come farlo...

All'inizio si scatena l'inferno, ma presto risolveremo questo apparente pasticcio.

Nota che in Eclipse, l'editor PyDev suggerisce problemi di codifica evidenziandoli in rosso nella parte destra dell'editor di codice.

Forse non dovremmo programmare in OOP dopotutto, ma questo è quello che facciamo, e per ottime ragioni:

```
GU_OOP_classes.py
...
60@ class OOP():
61
62     # Create instance
63     win = tk.Tk()
64
65     # Add a title
66     win.title("Python GUI")
67
68     tabControl = ttk.Notebook(win)          # Create Tab Control
69
70     tab1 = ttk.Frame(tabControl)           # Create a tab
71     tabControl.add(tab1, text='Tab 1')      # Add the tab
72     tab2 = ttk.Frame(tabControl)           # Add a second tab
73     tabControl.add(tab2, text='Tab 2')      # Make second tab visible
74
75     tabControl.pack(expand=1, fill="both") # Pack to make visible
76
77     # LabelFrame using tab1 as the parent
78     mighty = ttk.LabelFrame(tab1, text='Mighty Python.')
79     mighty.grid(column=0, row=0, padx=8, pady=4)
80
81     # Modify adding a Label using mighty as the parent instead of win
82     a_label = ttk.Label(mighty, text="Enter a name:")
83     a_label.grid(column=0, row=0, sticky='W')
84
85     # Modified Button Click Function
86     def click_me():
87         action.configure(text='Hello ' + name.get() + ' ' +
88                         number_chosen.get())
89
90     # Adding a Textbox Entry widget
91     name = tk.StringVar()
92     name_entered = ttk.Entry(mighty, width=12, textvariable=name)
```

Search Console PyUnit <terminated> C:\Eclipse_NEON_workspace\2nd Edition Python GUI Programming Cookbook\Ch04_Code\GUI_OOP_classes.py

Dobbiamo solo anteporre a tutte le variabili il se stesso parola chiave e associare anche le funzioni alla classe usando se stesso, che trasforma ufficialmente e tecnicamente le funzioni in metodi.



C'è una differenza tra funzioni e metodi. Python lo rende molto chiaro. I metodi sono associati a una classe mentre le funzioni no. Possiamo anche mescolare i due all'interno dello stesso modulo Python.

Mettiamo tutto come prefisso se stesso per eliminare il rosso in modo da poter eseguire nuovamente il nostro codice:

```
# Modified Button Click Function
def click_me(self):
    self.action.configure(text='Hello ' + self.name.get() + ' ' +
    self.number_chosen.get())
```

Una volta fatto questo per tutti gli errori evidenziati in rosso, possiamo eseguire nuovamente il nostro codice Python. Il cliccami la funzione è ora legata alla classe ed è diventata ufficialmente un metodo.

Sfortunatamente, iniziare in modo procedurale e poi tradurlo in OOP non è così semplice come ho affermato in precedenza. Il codice è diventato un enorme casino. Questo è un ottimo motivo per iniziare a programmare in Python utilizzando il modello di codifica OOP.



Python è bravo a fare le cose nel modo più semplice. Il codice facile spesso diventa più complesso (perché all'inizio era facile). Una volta che diventiamo troppo complessi, il refactoring del nostro codice procedurale in quello che potrebbe essere veramente un codice OOP diventa più difficile con ogni singola riga di codice.

Stiamo traducendo il nostro codice procedurale in codice orientato agli oggetti. Guardando tutti i problemi in cui ci siamo cacciati, tradurre solo 200+ righe di codice Python in OOP potrebbe suggerire che potremmo anche iniziare a codificare in OOP dall'inizio.

In realtà abbiamo interrotto alcune delle nostre funzionalità precedentemente funzionanti. Usando **Scheda 2** e fare clic sui pulsanti di opzione non funziona più. Dobbiamo rifattorizzare di più.

Il codice procedurale era facile nel senso che si trattava semplicemente di codifica dall'alto verso il basso. Ora che abbiamo inserito il nostro codice in una classe, dobbiamo spostare tutte le funzioni di callback nei metodi. Funziona, ma richiede un po' di lavoro per tradurre il nostro codice originale:

```
#####
# Il nostro codice procedurale assomigliava a questo:
#####
# Pulsante Clic Funzione
```

```
def click_me():
    action.configure(text='Hello ' + name.get() + ' ' +
                    numero_scelto.get())

# Aggiunta di un widget Voce casella di testo
nome = tk.StringVar()
name_entered = ttk.Entry(mighty, width=12, textvariable=name)
name_entered.grid(column=0, row=1, sticky='W')

# Aggiunta di un pulsante
action = ttk.Button(mighty, text="Click Me!", command=click_me)
action.grid(column=2, row=1)

ttk.Label(mighty, text="Scegli un numero:").grid(column=1, row=0) numero =
tk.StringVar()
numero_scelto = ttk.Combobox(potente, larghezza=12,
                             textvariable=numero, stato='sola lettura')
numero_scelto['valori'] = (1, 2, 4, 42, 100)
numero_scelto.grid(colonna=1, riga=1)
numero_scelto.corrente(0)

*****
Il nuovo codice OOP ha questo aspetto:
*****

classe OOP():
    def __init__(self):
        # Crea istanza
        self.win = tk.Tk()

        # Aggiungi un titolo
        self.win.title("GUI Python")
        self.create_widgets()

    # Pulsante di richiamata
    def click_me(self):
        self.action.configure(text='Hello ' + self.name.get() + ' '
                            + self.number_chosen.get())
        # ... più metodi di callback

    def create_widgets(self):
        # Crea controllo scheda
        tabControl = ttk.Notebook(self.win) tab1 =
        ttk.Frame(tabControl) tabControl.add(tab1,                 # Crea una scheda
                                                # Aggiungi la scheda
                                                # Crea una seconda scheda
                                                # Aggiungi seconda scheda
                                                # Pack per rendere visibile
        tabControl.pack (espandi = 1, riempি = "entrambi")
```

```
# Aggiunta di un widget Voce casella di testo - utilizzando self
self.name = tk.StringVar()
nome_inserito = ttk.Entry(potente, larghezza=12,
                           textvariable=self.name)
name_entered.grid(column=0, row=1, sticky='W')

# Aggiunta di un pulsante: utilizzo di sé
self.action = ttk.Button(mighty, text="Cliccami!",
                           comando=self.click_me)
self.action.grid(colonna=2, riga=1)
# ...
# =====
# Avvia GUI
# ===== oop = OOP() # crea un'istanza della classe

# usa la variabile di istanza per chiamare mainloop tramite win
oop.win.mainloop()
```

Abbiamo spostato i metodi di callback all'inizio del modulo, all'interno della nuova classe OOP. Abbiamo spostato tutto il codice di creazione del widget in un metodo piuttosto lungo, che chiamiamo nell'inizializzatore della classe.

Tecnicamente, in fondo al cofano del codice di basso livello, Python ha un costruttore, eppure Python ci libera da ogni preoccupazione al riguardo. È curato per noi.

Invece, oltre a un vero costruttore, Python ci fornisce un inizializzatore.

Siamo fortemente incoraggiati a utilizzare questo inizializzatore. Possiamo usarlo per passare i parametri alla nostra classe, inizializzando le variabili che vogliamo usare all'interno della nostra istanza di classe.



In Python, possono esistere diverse classi all'interno dello stesso modulo Python.

A differenza di Java, che ha una convenzione di denominazione molto rigida (senza la quale non funziona), Python è molto più flessibile.



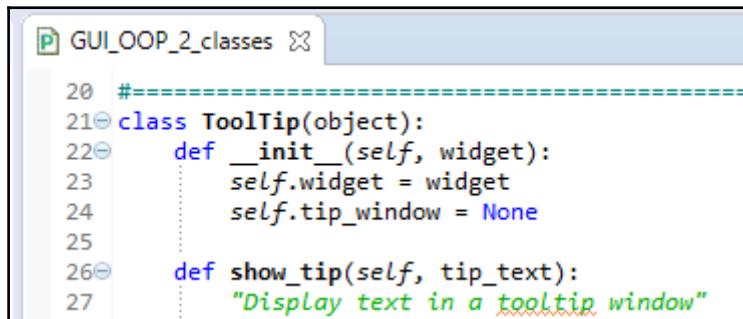
Possiamo creare più classi all'interno dello stesso modulo Python. A differenza di Java, non dipendiamo da un nome di file che deve corrispondere a ciascun nome di classe. Python spacca davvero!

Una volta che la nostra GUI Python diventa grande, suddivideremo alcune classi nei propri moduli ma, a differenza di Java, non sarà necessario. In questo libro e progetto, manterremo alcune classi nello stesso modulo mentre, allo stesso tempo, suddivideremo altre classi nei rispettivi moduli, importandole in quello che può essere considerato un principale() funzione (questo non è C, ma possiamo pensare in modo C perché Python è molto flessibile).

Quello che abbiamo ottenuto finora è aggiungere il Descrizione comando class al nostro modulo Python e refactoring del nostro codice Python procedurale in codice Python OOP.

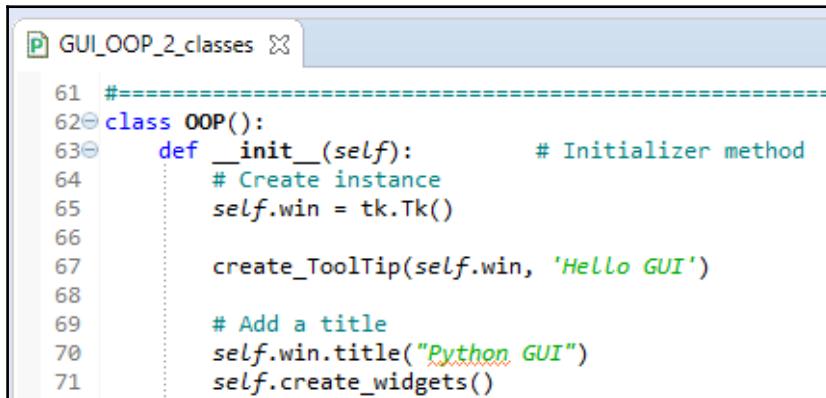
Qui, in questa ricetta, possiamo vedere che più di una classe può vivere nello stesso modulo Python.

Roba bella, davvero!



```
20 =====
21 class ToolTip(object):
22     def __init__(self, widget):
23         self.widget = widget
24         self.tip_window = None
25
26     def show_tip(self, tip_text):
27         "Display text in a tooltip window"
```

Entrambi i Descrizione comando classe e il OOP class risiedono all'interno dello stesso modulo Python:



```
61 =====
62 class OOP():
63     def __init__(self):          # Initializer method
64         # Create instance
65         self.win = tk.Tk()
66
67         create_ToolTip(self.win, 'Hello GUI')
68
69         # Add a title
70         self.win.title("Python GUI")
71         self.create_widgets()
```

Come funziona...

In questa ricetta, abbiamo avanzato il nostro codice procedurale in codice OOP. Python ci consente di scrivere codice sia in uno stile pratico che procedurale come il linguaggio di programmazione C. Allo stesso tempo, abbiamo la possibilità di codificare in uno stile OOP, come Java, C# e C++.

Scrivere funzioni di callback

All'inizio, le funzioni di callback possono sembrare un po' intimidatorie. Chiami la funzione, passandole alcuni argomenti, e poi la funzione ti dice che è davvero molto occupata e ti richiamerà!

Ti chiedi: questa funzione mi richiamerà mai? E quanto devo aspettare? In Python, anche le funzioni di callback sono facili e, sì, di solito ti richiamano. Devono solo completare prima il compito loro assegnato (ehi, sei stato tu a codificarli per primo...).

Cerchiamo di capire un po' di più su cosa succede quando codifichiamo callback nella nostra GUI. La nostra GUI è guidata dagli eventi. Dopo che è stato creato e visualizzato sullo schermo, di solito rimane lì in attesa che si verifichi un evento. Sostanzialmente sta aspettando che gli venga inviato un evento. Possiamo inviare un evento alla nostra GUI facendo clic su uno dei suoi pulsanti di azione. Questo crea un evento e, in un certo senso, abbiamo chiamato la nostra GUI inviandogli un messaggio.

Ora, cosa dovrebbe succedere dopo aver inviato un messaggio alla nostra GUI? Ciò che accade dopo aver fatto clic sul pulsante dipende dal fatto che abbiamo creato un gestore di eventi e l'abbiamo associato a questo pulsante. Se non abbiamo creato un gestore di eventi, fare clic sul pulsante non avrà alcun effetto. Il gestore di eventi è una funzione di callback (o metodo, se usiamo le classi). Il metodo di callback è anche seduto lì passivamente, come la nostra GUI, in attesa di essere invocato. Una volta che la nostra GUI ha fatto clic sul pulsante, invocherà il callback.

Il callback spesso esegue alcune elaborazioni e, una volta terminato, restituisce il risultato alla nostra GUI.



In un certo senso, possiamo vedere che la nostra funzione di callback richiama la nostra GUI.

Prepararsi

L'interprete Python esegue tutto il codice in un modulo una volta, trovando eventuali errori di sintassi e segnalandoli. Non puoi eseguire il tuo codice Python se non hai la sintassi corretta. Ciò include l'indentazione (se non risulta in un errore di sintassi, l'indentazione errata di solito provoca un bug).

Al successivo giro di analisi, l'interprete interpreta il nostro codice e lo esegue.

In fase di esecuzione, possono essere generati molti eventi della GUI e di solito sono le funzioni di callback che aggiungono funzionalità ai widget della GUI.

Come farlo...

Ecco il richiamo per il Spinbox aggiaggio:

```
37     # Spinbox callback
38     def _spin(self):
39         value = self.spin.get()
40         print(value)
41         self.scroll.insert(tk.END, value + '\n')
42
43     # Adding a Spinbox widget
44     self.spin = Spinbox(mighty, values=(1, 2, 4, 42, 100), width=5, bd=9, command=self._spin)
45     self.spin.grid(column=0, row=2)
```

Come funziona...

Abbiamo creato un metodo di callback nel OOP classe che viene chiamata quando selezioniamo un valore dal Spinbox widget perché abbiamo associato il metodo al widget tramite il comando discussione (comando=self._spin). Usiamo un carattere di sottolineatura iniziale per suggerire il fatto che questo metodo deve essere rispettato come un metodo Java privato.

Python evita intenzionalmente le restrizioni linguistiche, come private, public, friend e così via. In Python, invece, usiamo le convenzioni di denominazione. Ci si aspetta che i doppi underscore iniziali e finali che circondano una parola chiave siano limitati al linguaggio Python e ci si aspetta che non li usino nel nostro codice Python.

Tuttavia, possiamo usare un prefisso di sottolineatura iniziale con un nome di variabile o una funzione per fornire un suggerimento che questo nome deve essere rispettato come aiutante privato.

Allo stesso tempo, possiamo postfiggere un singolo carattere di sottolineatura se desideriamo usare quelli che altrimenti sarebbero nomi Python incorporati. Ad esempio, se volessimo abbreviare la lunghezza di una lista, potremmo fare quanto segue:

```
len_ = len(aLista)
```



Spesso, il carattere di sottolineatura è difficile da leggere e facile da controllare, quindi questa potrebbe non essere l'idea migliore in pratica.

Creazione di componenti GUI riutilizzabili

Creeremo componenti GUI riutilizzabili usando Python. In questa ricetta, la manterremo semplice spostando il nostro comando classe nel proprio modulo. Quindi, lo importeremo e lo utilizzeremo per visualizzare i suggerimenti su diversi widget della nostra GUI.

Prepararsi

Stiamo costruendo il nostro codice da Capitolo 3, *Personalizzazione dell'aspetto: GUI_tooltip.py*.

Come farlo...

Inizieremo col rompere il nostro Descrizione comando classe in un modulo Python separato. Lo migliorneremo leggermente per passare il widget di controllo e il testo del suggerimento che desideriamo visualizzare quando passiamo il mouse sul controllo.

Creiamo un nuovo modulo Python e posizioniamo il Descrizione comando codice della classe in esso e quindi importare questo modulo nel nostro modulo principale.

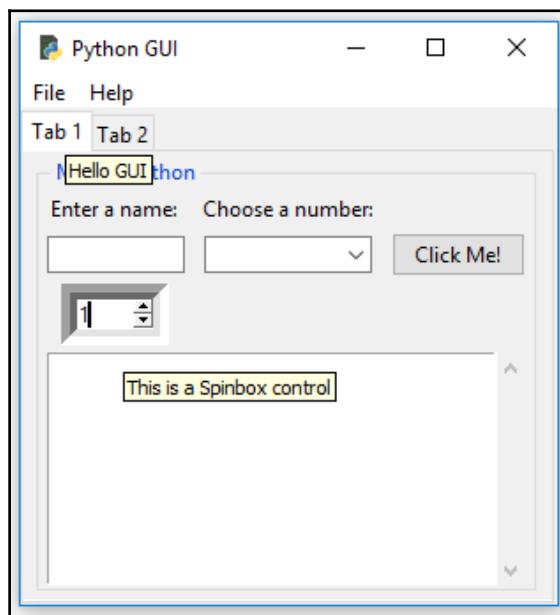
Quindi riutilizziamo l'importato Descrizione comando class creando diversi tooltip, che possono essere visualizzati quando si passa il mouse su diversi dei nostri widget della GUI.

Refactoring il nostro comune Descrizione comando il codice di classe nel proprio modulo ci aiuta a riutilizzare questo codice da altri moduli. Invece di copiare/incollare/modificare, utilizziamo il principio DRY e il nostro codice comune si trova in una sola posizione, quindi quando modifichiamo il codice, tutti i moduli che lo importano riceveranno automaticamente l'ultima versione del nostro modulo.



ASCIUTTO sta per **Non ripeterti**, e lo vedremo di nuovo in un capitolo successivo. Possiamo fare cose simili girando il nostro**Scheda 3** l'immagine in un componente riutilizzabile. Per mantenere semplice il codice di questa ricetta, abbiamo rimosso**Scheda 3**, ma puoi sperimentare con il codice del capitolo precedente. Il codice viene riutilizzato da `GUI_tooltip.py`.

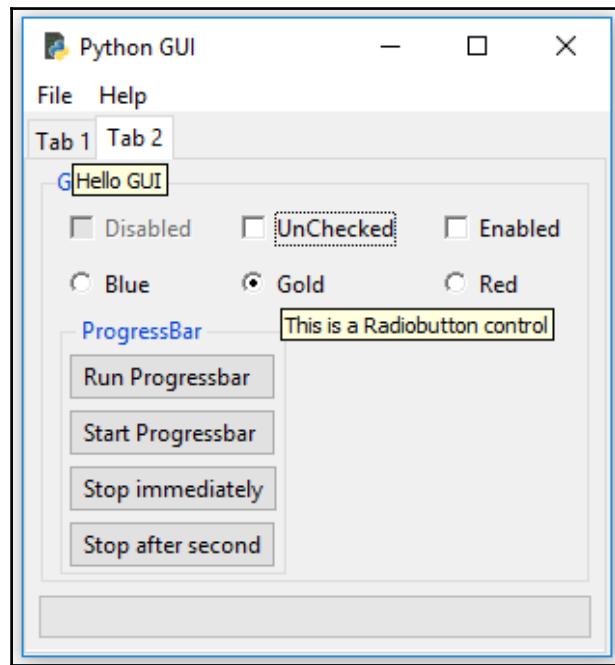
Considera di correre `GUI_tooltip.py`:



Considera il seguente codice:

```
# Aggiungi un suggerimento alla casella numerica Spin  
tt.create_ToolTip(self.spin, 'Questo è un controllo Spinbox')  
  
# Aggiungi suggerimenti a più widget  
tt.create_ToolTip(self.name_entered, 'Questo è un controllo Entry')  
tt.create_ToolTip(self.action, 'Questo è un controllo Button')  
tt.create_ToolTip(self.scrol, 'Questo è un controllo ScrolledText')
```

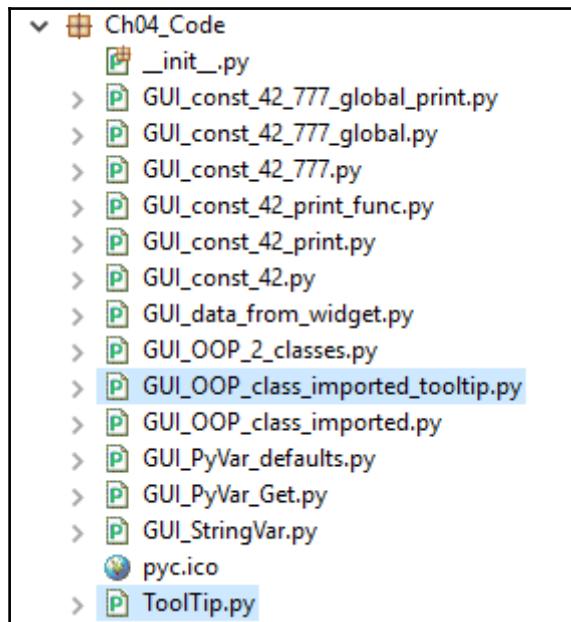
Funziona anche sulla seconda scheda:



Le istruzioni di importazione hanno il seguente aspetto:

```
GUI_OOP_class_imported_tooltip □ P ToolTip
13 from tkinter import messagebox as msg
14 from tkinter import Spinbox
15 from time import sleep
16 import Ch04_Code.ToolTip as tt
17
18 GLOBAL_CONST = 42
19
20 =====
21 class OOP():
22     def __init__(self):          # Initializer method
23         # Create instance
24         self.win = tk.Tk()
25
26         tt.create_ToolTip(self.win, 'Hello GUI')
27
```

La nuova struttura del codice ora si presenta così:



E il codice rotto (noto anche come refactoring) in un modulo separato assomiglia a questo:

```
GUI_OOP_class_imported_tooltip
ToolTip
9 import tkinter as tk
10
11 =====
12 class ToolTip(object):
13     def __init__(self, widget):
14         self.widget = widget
15         self.tip_window = None
16
17     def show_tip(self, tip_text):
18         "Display text in a tooltip window"
```

Come funziona...

Negli screenshot precedenti, possiamo vedere diversi messaggi di tooltip visualizzati. Quello per la finestra principale potrebbe sembrare un po' fastidioso, quindi è meglio non visualizzare un tooltip per la finestra principale, perché vogliamo davvero evidenziare le funzionalità dei singoli widget. Il modulo della finestra principale ha un titolo che ne spiega lo scopo; non c'è bisogno di un tooltip.

5

Grafici Matplotlib

In questo capitolo, creeremo bellissimi grafici usando Python 3.6 con il Matplotlib modulo. Tratteremo le seguenti ricette:

- Creazione di bellissimi grafici utilizzando Matplotlib
- Installazione Matplotlib usando pip e quando extension Creare
- il nostro primo grafico
- Posizionamento di etichette sui grafici
- Come dare al grafico una legenda
- Grafici di scala
- Regolazione dinamica della scala dei grafici

introduzione

In questo capitolo creeremo bellissimi grafici che rappresentano visivamente i dati. A seconda del formato dell'origine dati, possiamo tracciare una o più colonne di dati nello stesso grafico.

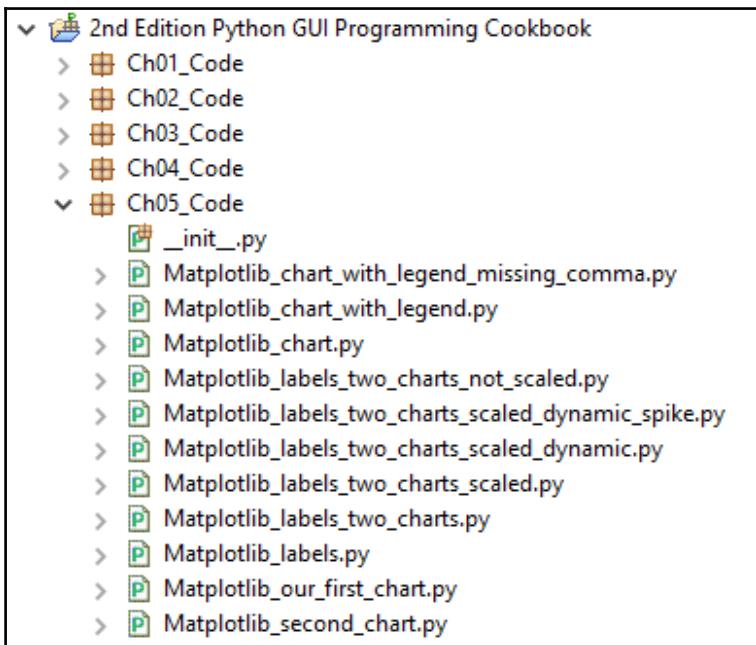
Useremo Python Matplotlib modulo per creare i nostri grafici.

Per creare questi grafici, dobbiamo scaricare moduli Python aggiuntivi e ci sono diversi modi per installarli.

Questo capitolo spiegherà come scaricare il Matplotlib Modulo Python insieme a tutti gli altri moduli Python richiesti e i modi per farlo.

Dopo aver installato i moduli richiesti, creeremo i nostri grafici Pythonic.

Ecco la panoramica dei moduli Python per questo capitolo:



Creare bellissimi grafici usando Matplotlib

Questa ricetta ci introduce al Matplotlib Modulo Python, che ci consente di creare grafici visivi utilizzando Python 3.6 e versioni successive.

L'URL, <http://matplotlib.org/users/screenshots.html>, è un ottimo punto di partenza esplorando il mondo di Matplotlib, e ci insegna come creare molti grafici che non sono presentati in questo capitolo.

Prepararsi

Per utilizzare il Matplotlib Python, dobbiamo prima installare questo modulo e molti altri moduli Python correlati.

Se stai eseguendo una versione di Python precedente alla 3.6, ti incoraggio ad aggiornare la tua versione, poiché utilizzeremo Python pipì module in questo capitolo per installare i moduli Python richiesti e pipì è installato con 3.6 e versioni successive.

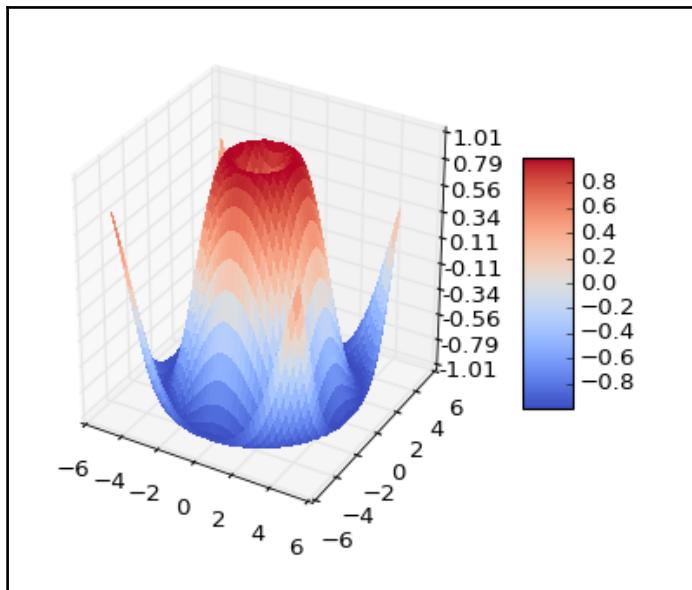


È possibile installare pipì con le versioni precedenti di Python 3 ma il processo non è molto intuitivo, quindi è decisamente meglio aggiornare alla 3.6 o superiore.

Come farlo...

L'immagine seguente è un esempio di quali incredibili grafici possono essere creati usando Python con il Matplotlib modulo:

Matplotlib_chart.py



Nel seguente frammento di codice, ho copiato del codice dal <http://matplotlib.org/> sito web, che crea questo incredibile grafico. Ho anche aggiunto alcuni commenti al codice. Ci sono molti esempi disponibili su questo sito e ti incoraggio a provarli finché non trovi il tipo di grafici che desideri creare.

Ecco il codice per creare il grafico in meno di 25 righe di codice Python, inclusi gli spazi bianchi:

```
da mpl_toolkits.mplot3d import Axes3D da
matplotlib importa cm
da matplotlib.ticker import LinearLocator, FormatStrFormatter import
matplotlib.pyplot as plt
importa numpy come np

fig = plt.figure()                                     # crea una figura
ax = fig.gca(proiezione='3d') X =                   # crea un asse tridimensionale
np.arange(-5, 5, 0.25) Y = np.arange(-5,             # intervallo orizzontale
5, 0.25)                                         # intervallo verticale
X, Y = np.meshgrid(X, Y) R =                      # crea una griglia speciale
np.sqrt(X**2 + Y**2) Z =                          # calcola la radice quadrata
np.sin(R)                                           # calcola seno

# # usa #@UndefinedVariable sotto per ignorare l'errore per cm.coolwarm
surf = ax.plot_surface(X, Y, Z, rstride=1, cstride=1,
                       cmap=cm.coolwarm, linewidth=0,
                       antialiased=False)
ax.set_zlim(-1.01, 1.01)                            # l'asse z è la terza dimensione

ax.xaxis.set_major_locator(LinearLocator(10))
ax.xaxis.set_major_formatter(FormatStrFormatter('%0.02f'))

fig.colorbar(surf, riduci=0,5, aspetto=5)

plt.mostra()                                         # mostra la figura
```



L'esecuzione del codice utilizzando Python 3.6 o versioni successive con il plug-in Eclipse PyDev potrebbe mostrare alcuni errori di importazione e variabili non risolti. Puoi semplicemente disabilitarli in Eclipse tramite #@Importazione non risolta e
@Variabile non definita.

Ignora questi errori se stai sviluppando utilizzando Eclipse, poiché il codice verrà eseguito correttamente.

Come funziona...

Per creare bellissimi grafici, come mostrato in precedenza, dobbiamo scaricare Matplotlib così come molti altri moduli Python.

La seguente ricetta ci guiderà attraverso come scaricare e installare con successo tutti i moduli richiesti che ci permetteranno di creare i nostri bellissimi grafici.

Installazione di Matplotlib usando pip con estensione whl

Il modo usuale per scaricare moduli Python aggiuntivi è usare pip. Il pip viene preinstallato con l'ultima versione di Python (3.6 e successive).



Se stai usando una versione precedente di Python, potresti dover scaricare entrambi pip e strumenti di configurazione te stesso.

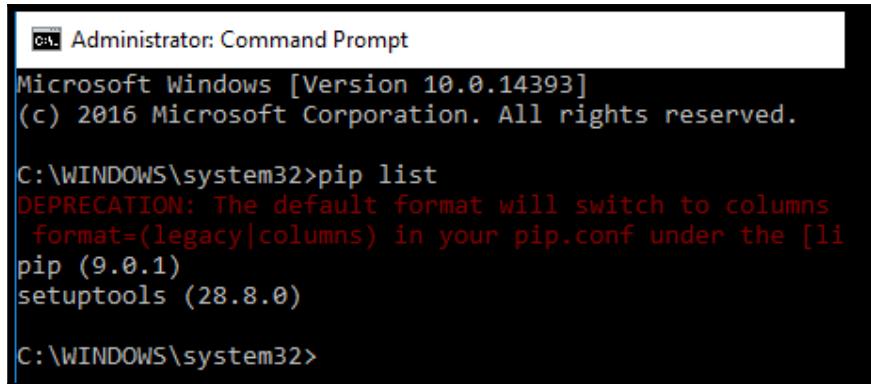
Questa ricetta mostrerà come installare con successo Matplotlib usando pip. Utilizzeremo il .whl estensione per questa installazione, quindi questa ricetta ti mostrerà anche come installare il ruota modulo.

Prepararsi

Per prima cosa, scopriamo se hai il ruota modulo già installato. Il ruota modulo è necessario per scaricare e installare i pacchetti Python con estensione .whl.

Possiamo scoprire quali moduli abbiamo attualmente installato utilizzando pip.

Dal prompt dei comandi di Windows, eseguire il lista di semi comando:

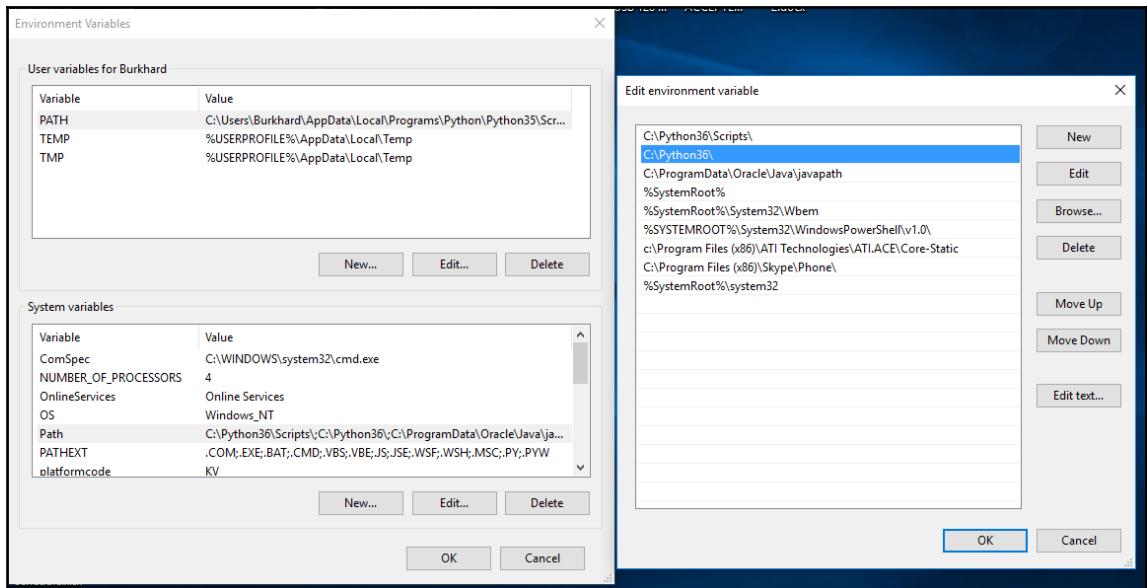


```
c:\ Administrator: Command Prompt
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32>pip list
DEPRECATION: The default format will switch to columns
format=(legacy|columns) in your pip.conf under the [li
pip (9.0.1)
setuptools (28.8.0)

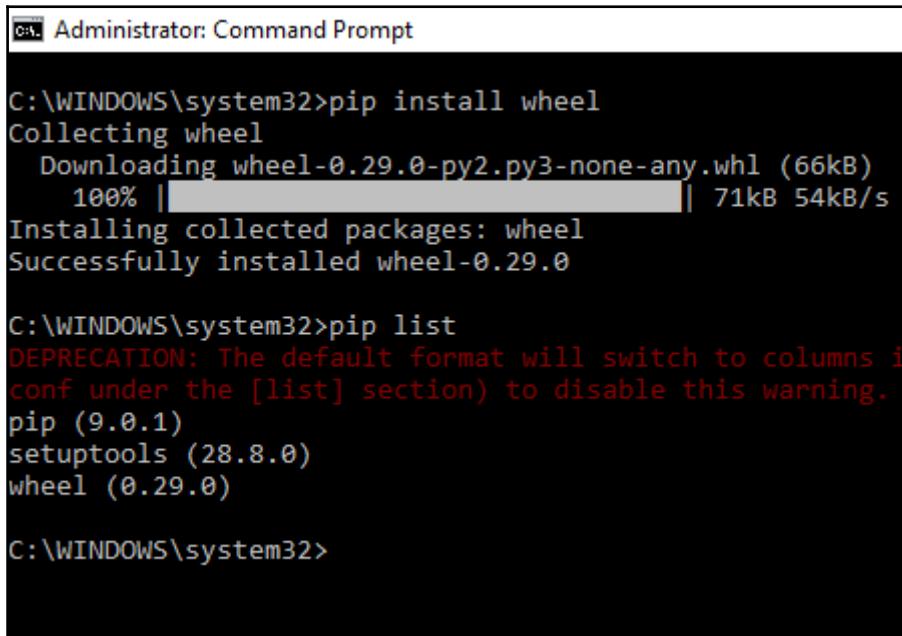
C:\WINDOWS\system32>
```

Se ricevi un errore durante l'esecuzione di questo comando, potresti voler controllare se Python è sul tuo percorso ambientale. Se attualmente non lo è, aggiungilo a **Variabili di sistema | Sentiero** (in basso a sinistra) cliccando su **Modificare...** pulsante. Quindi, fare clic su **Nuovo** pulsante (in alto a destra) e digita il percorso della tua installazione di Python. Inoltre, aggiungi la directory Scripts, come pip.exe vive là:



Se hai installato più di una versione di Python, è una buona idea spostare Python 3.6 in cima all'elenco. Quando scriviamo pip install <modulo>, la prima versione trovata in **Variabili di sistema** | **Sentiero** potrebbe essere utilizzata e potresti ricevere alcuni errori imprevisti se una versione precedente di Python si trova sopra Python 3.6.

Corriamo ruota di installazione pip e quindi verificare se è stato installato correttamente utilizzando pip in elenco:



```
C:\WINDOWS\system32>pip install wheel
Collecting wheel
  Downloading wheel-0.29.0-py2.py3-none-any.whl (66kB)
    100% |██████████| 71kB 54kB/s
Installing collected packages: wheel
Successfully installed wheel-0.29.0

C:\WINDOWS\system32>pip list
DEPRECATION: The default format will switch to columns in
conf under the [list] section) to disable this warning.
pip (9.0.1)
setuptools (28.8.0)
wheel (0.29.0)

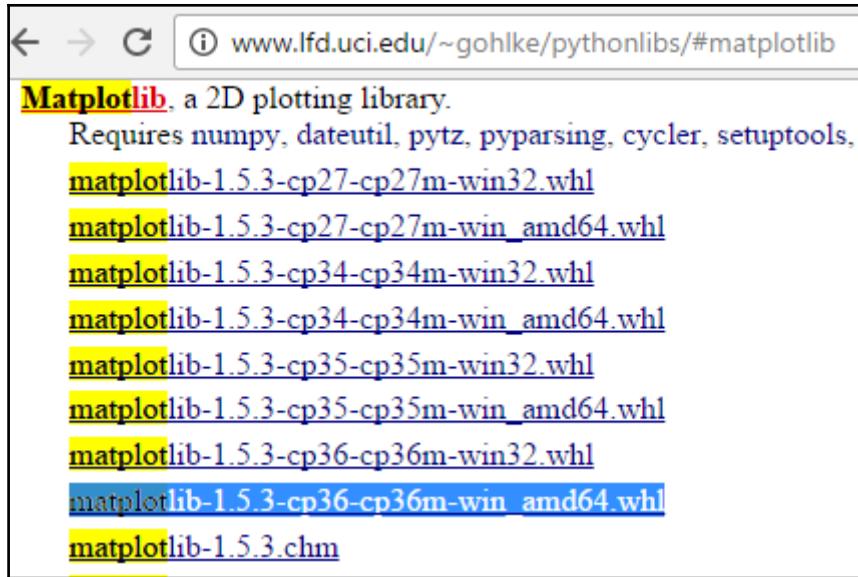
C:\WINDOWS\system32>
```



Se sei davvero molto abituato a Python 2.7 e insisti per eseguire il codice in Python 2.7, puoi provare questo trucco. Dopo che tutto funziona con Python 3.6, puoi rinominare il 3.6 python.exe per python3.exe e poi divertiti a usare sia 2.7 che 3.6 digitando python.exe o python3.exe in una finestra di comando per eseguire i diversi eseguibili Python. È un trucco. Se vuoi davvero percorrere questa strada, sei da solo, ma può funzionare.

Come farlo...

Con il modulo ruota installato, ora possiamo scaricare e installare Matplotlib a partire dal <http://www.lfd.uci.edu/~gohlke/pythonlibs/#matplotlib>:



Assicurati di scaricare e installare il Matplotlib versione che corrisponde alla versione di Python che stai utilizzando. Ad esempio, scarica e installa [Matplotlib-1.5.3-cp36-cp36m-win-amd64.whl](#) se hai Python 3.6 installato su un sistema operativo a 64 bit, come Microsoft Windows 10.



Il **amd64** nel mezzo del nome dell'eseguibile significa che stai installando la versione a 64 bit. Se stai utilizzando un sistema x86 a 32 bit, installa **amd64** non funzionerà. Problemi simili possono verificarsi se hai installato una versione a 32 bit di Python e scaricato moduli Python a 64 bit.

Dopo aver scaricato il programma di installazione della ruota, ora possiamo usare pip. A seconda di cosa hai già installato sul tuo sistema, eseguendo il `pip install Matplotlib-1.5.3cp36-cp36m-win-amd64.whl` il comando potrebbe avviarsi correttamente, ma potrebbe non essere eseguito fino al completamento. Ecco uno screenshot di ciò che potrebbe accadere durante l'installazione:

```
C:\Windows\system32\cmd.exe

C:\Users\Burkhard\Desktop\2nd EDITION PACKT PYTHON GUI COOKBOOK\SW DOWNLOADS>pip install matplotlib-1.5.3-cp36-cp36m-win_amd64.whl
Processing c:\users\burkhard\desktop\2nd edition packt python gui cookbook\sw downloads\matplotlib-1.5.3-cp36-cp36m-win_amd64.whl
Collecting cypher (from matplotlib==1.5.3)
  Downloading cypher-0.10.0-py2.py3-none-any.whl
Collecting python-dateutil (from matplotlib==1.5.3)
  Downloading python_dateutil-2.6.0-py2.py3-none-any.whl (194kB)
    100% |██████████| 194kB 728kB/s
Collecting pytz (from matplotlib==1.5.3)
  Downloading pytz-2016.7-py2.py3-none-any.whl (480kB)
    100% |██████████| 481kB 546kB/s
Collecting pyparsing!=2.0.0,!=2.0.4,!=2.1.2,>=1.5.6 (from matplotlib==1.5.3)
  Downloading pyparsing-2.1.10-py2.py3-none-any.whl (56kB)
    100% |██████████| 61kB 491kB/s
Collecting numpy>=1.6 (from matplotlib==1.5.3)
  Downloading numpy-1.11.2.tar.gz (4.2MB)
    100% |██████████| 4.2MB 109kB/s                                         Collecting six (from cypher->matplotlib==1.5.3)

  Downloading six-1.10.0-py2.py3-none-any.whl
Building wheels for collected packages: numpy
  Running setup.py bdist_wheel for numpy ... error

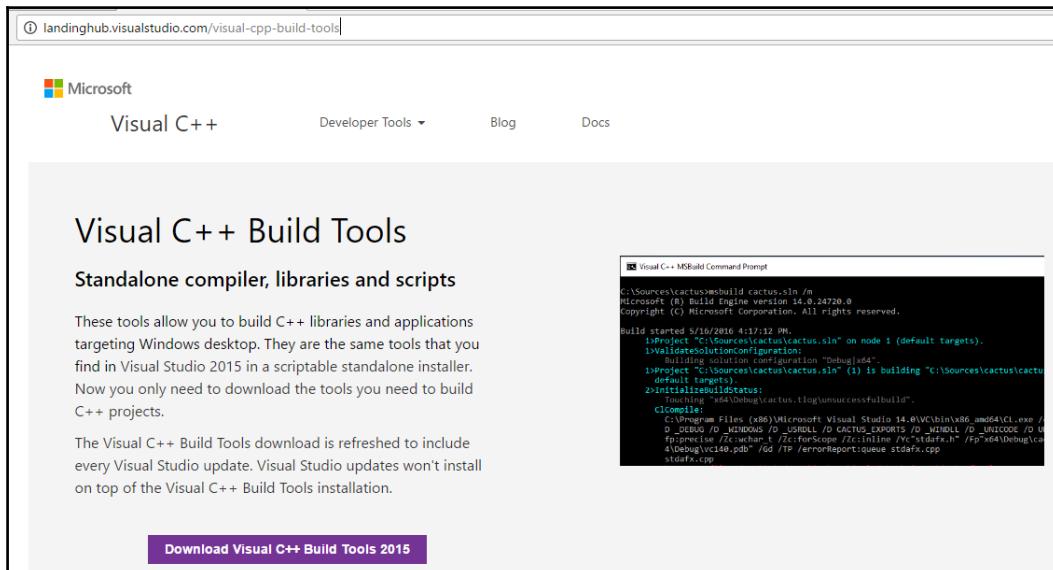
  error: Microsoft Visual C++ 14.0 is required. Get it with "Microsoft Visual C++ Build Tools": http://landinghub.visualstudio.com/visual-cpp-build-tools

build_src
building_py_modules_sources
building_library_npymath_sources
No module named 'numpy.distutils._msvccompiler' in numpy.distutils; trying from distutils
error: Microsoft Visual C++ 14.0 is required. Get it with "Microsoft Visual C++ Build Tools": http://landinghub.visualstudio.com/visual-cpp-build-tools

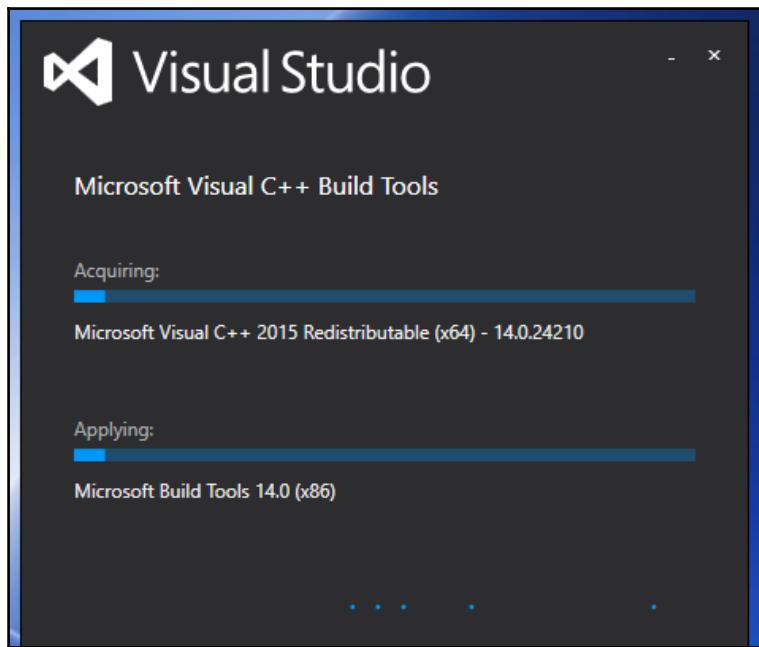
Command "c:\python36\python.exe -c "import setuptools, tokenize; f = open('c:\\users\\burkhard\\appdata\\local\\temp\\pip-build-hdm04rl\\numpy\\\\setup.py', encoding='utf8'); code=f.read().replace('\\r', '\\n'); f.close(); exec(compile(code, f.name, 'exec'))" install --record c:\\users\\burkhard\\appdata\\local\\temp\\pip-nugm4en_record\\install-record.txt --single-version-externally-managed --compile" failed with error code 1 in c:\\users\\burkhard\\appdata\\local\\temp\\pip-build-hdm04rl\\numpy\\"

C:\Users\Burkhard\Desktop\2nd EDITION PACKT PYTHON GUI COOKBOOK\SW DOWNLOADS
```

L'installazione ha riscontrato un errore. Il modo per risolvere questo problema è scaricare e installare i **Strumenti di compilazione di Microsoft Visual C++**, e lo facciamo dal sito web menzionato in la schermata precedente (<http://landinghub.visualstudio.com/visual-cpp-build-tools>):



L'avvio dell'installazione di MS VC++ Build Tools si presenta come segue:



Dopo aver installato con successo gli strumenti di compilazione, ora possiamo eseguire il nostro Matplotlib installazione a completamento. Quindi, digita lo stesso pip install comando che abbiamo usato prima:

```
Administrator: Command Prompt
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

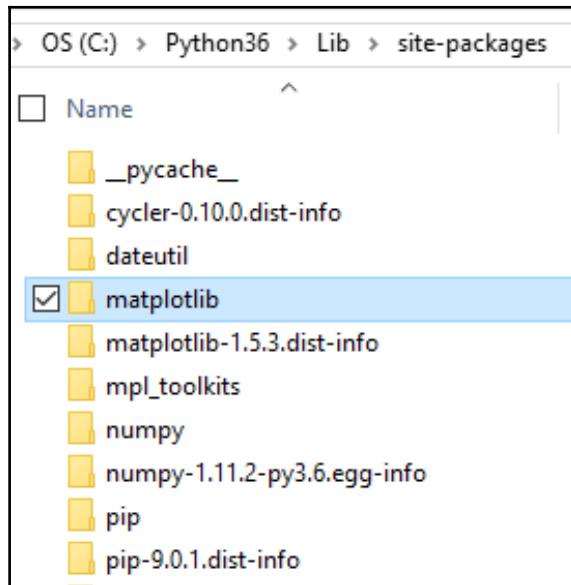
C:\WINDOWS\system32>cd C:\Users\Burkh\Desktop\2nd EDITION PACKT PYTHON GUI COOKBOOK\SW DOWNLOADS

C:\Users\Burkh\Desktop\2nd EDITION PACKT PYTHON GUI COOKBOOK\SW DOWNLOADS>pip install matplotlib-1.5.3-cp36-cp36m-win_amd64.whl
Processing c:\users\burkh\desktop\2nd edition packt python gui cookbook\sw downloads\matplotlib-1.5.3-cp36-cp36m-win_amd64.whl
Requirement already satisfied: python-dateutil in c:\python36\lib\site-packages (from matplotlib==1.5.3)
Collecting cycler (from matplotlib==1.5.3)
  Using cached cycler-0.10.0-py3-none-any.whl
Collecting pytz (from matplotlib==1.5.3)
  Using cached pytz-2016.7-py2.py3-none-any.whl
Collecting pyparsing!=2.0.0,>=2.0.4,!=2.1.2,>=1.5.6 (from matplotlib==1.5.3)
  Using cached pyparsing-2.1.10-py2.py3-none-any.whl
Collecting numpy>=1.6 (from matplotlib==1.5.3)
  Using cached numpy-1.11.2.tar.gz
Requirement already satisfied: six>=1.5 in c:\python36\lib\site-packages (from python-dateutil->matplotlib==1.5.3)
Installing collected packages: cycler, pytz, pyparsing, numpy, matplotlib
  Running setup.py install for numpy ... done
Successfully installed cycler-0.10.0 matplotlib-1.5.3 numpy-1.11.2 pyparsing-2.1.10 pytz-2016.7

C:\Users\Burkh\Desktop\2nd EDITION PACKT PYTHON GUI COOKBOOK\SW DOWNLOADS>
```

Possiamo verificare di aver installato con successo Matplotlib guardando la nostra directory di installazione di Python. Al termine dell'installazione, il Matplotlib la cartella viene aggiunta a siti-pacchetti. A seconda di dove abbiamo installato Python, il percorso completo della cartella dei pacchetti del sito su Windows può essere:

C:\Python36\Lib\pacchetti-sito



Se vedi il matplotlib cartella aggiunta al pacchetti-sito cartella nella tua installazione di Python, quindi abbiamo installato con successo Matplotlib.

Come funziona...

Il modo comune per scaricare i moduli Python è usare pipa, mostrato in precedenza. Per installare tutti i moduli che Matplotlib richiede, il formato di download del sito Web principale da cui è possibile scaricarli è cambiato, utilizzando a quando formato.



Installare i moduli Python usando pipè di solito è molto facile. Eppure potresti incorrere in alcuni problemi imprevisti. Segui i passaggi precedenti e l'installazione avrà esito positivo.

Creare il nostro primo grafico

Ora che abbiamo installato tutti i moduli Python richiesti, possiamo creare i nostri grafici usando Matplotlib.

Possiamo creare grafici da poche righe di codice Python.

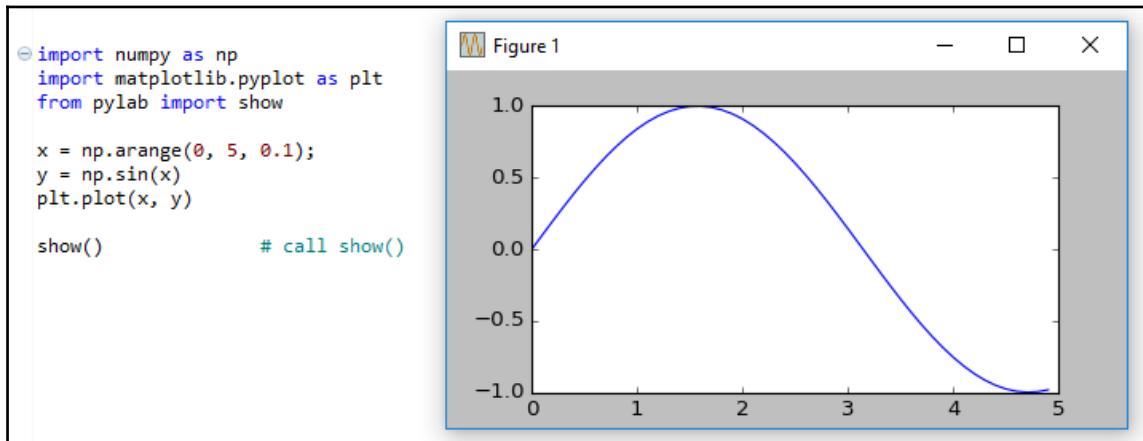
Prepararsi

Installazione riuscita Matplotlib, come mostrato nella ricetta precedente, è un requisito per questa ricetta.

Come farlo...

Utilizzando la quantità minima di codice, come presentato sul funzionario Matplotlib sito web, possiamo creare il nostro primo grafico. Be 'quasi. Il codice di esempio mostrato sul sito Web non funziona finché non importiamo ilmostrare funzione da pilab e poi chiamalo:

Matplotlib_our_first_chart.py



The image shows a Jupyter Notebook cell containing Python code and its resulting visualization. On the left, the code is displayed:

```
import numpy as np
import matplotlib.pyplot as plt
from pylab import show

x = np.arange(0, 5, 0.1);
y = np.sin(x)
plt.plot(x, y)

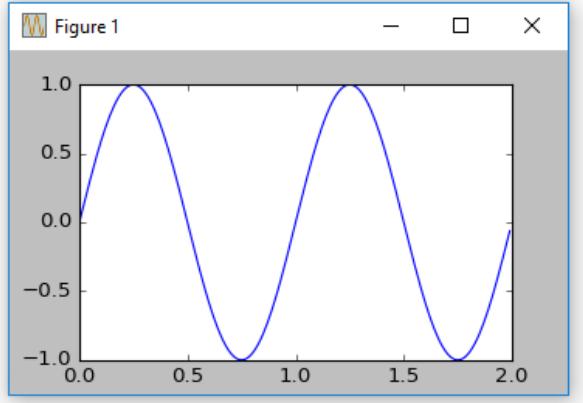
show()      # call show()
```

On the right, the resulting plot is shown in a window titled "Figure 1". The plot displays a blue sine wave curve. The x-axis ranges from 0 to 5 with major ticks at 0, 1, 2, 3, 4, and 5. The y-axis ranges from -1.0 to 1.0 with major ticks at -1.0, -0.5, 0.0, 0.5, and 1.0. The curve starts at (0,0), reaches a maximum of 1.0 at approximately x=1.57, crosses the x-axis at x=pi, reaches a minimum of -1.0 at approximately x=3.14, and returns to 0 at x=5.

Possiamo semplificare il codice e persino migliorarlo utilizzando un altro dei tanti esempi forniti sul sito ufficiale Matplotlib sito web. Il pilab il modulo è dotato di una propria funzione di stampa, quindi non è necessario importare matplotlib, dopotutto, se vogliamo semplificare il codice:

Matplotlib_second_chart.py

```
from pylab import show, arange, sin, plot, pi  
  
t = arange(0.0, 2.0, 0.01)  
s = sin( 2 * pi * t )  
plot(t, s)  
  
show()
```



Come funziona...

il pitone Matplotlib modulo, combinato con componenti aggiuntivi come insensibile, crea un ambiente di programmazione molto ricco che ci consente di eseguire calcoli matematici e tracciarli in grafici visivi molto facilmente.

Il arrangiare metodo di insensibile non intende organizzare nulla. Significa creare un intervallo, che viene utilizzato al posto del built-in di Python gamma operatore. Ilspazio di lino metodo può creare una confusione simile. Chi è/in e in cosa? spazio?

A quanto pare, il nome significa *vettore spaziato lineare*.

Il porcellino funzione mostrare mostra il grafico che abbiamo creato. chiamandomostrare() ha alcuni effetti collaterali quando si tenta di tracciare un altro grafico dopo aver creato con successo il primo.

Posizionamento di etichette sui grafici

Finora, abbiamo usato l'impostazione predefinita Matplotlib GUI. Ora, ne creeremo alcune utilizzando GUI che utilizzano Matplotlib.

Ciò richiederà qualche riga in più di codice Python e l'importazione di alcune librerie in più, e ne vale la pena, perché stiamo guadagnando il controllo dei nostri dipinti usando le tele.

Posizioniamo le etichette sia sull'asse orizzontale che su quello verticale, ovvero, X e y . Lo faremo creando un Matplotlib figura su cui disegneremo.

Imparerai anche come usare le sottotrame, che ti permetteranno di disegnare più di un grafico nella stessa finestra.

Prepararsi

Con i moduli Python necessari installati e sapendo dove trovare la documentazione online ufficiale e i tutorial, ora possiamo continuare con la nostra creazione di Matplotlib grafici.

Come farlo...

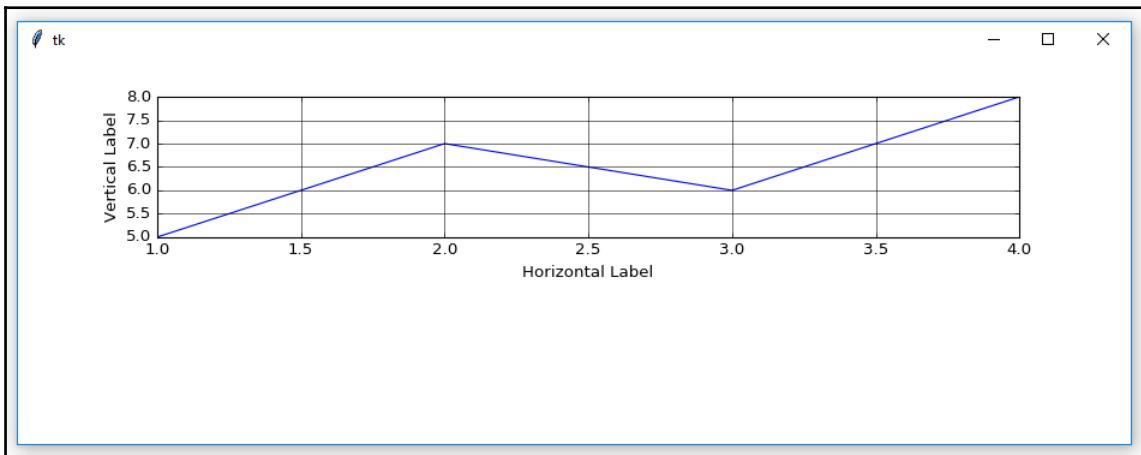
Mentre tracciare è il modo più semplice per creare un Matplotlib grafico, usando figura in combinazione con Tela crea un grafico più personalizzato, che ha un aspetto molto migliore e ci consente anche di aggiungere pulsanti e altri widget:

Matplotlib_labels.py

```
da matplotlib.figure import Figura
da matplotlib.backends.backend_tkagg import FigureCanvasTkAgg import tkinter as
tk
# -----
fig = Figure(figsize=(12, 8), facecolor='bianco')
# -----
# axis = fig.add_subplot(111) # 1 riga, 1 colonna, solo grafico
asse = fig.add_subplot(211) # 2 righe, 1 colonna, grafico in alto
# -----
xValori = [1,2,3,4]
yValues = [5,7,6,8]
axis.plot(xValues, yValues)
axis.set_xlabel('Horizontal Label')
axis.set_ylabel('Vertical Label')
# axis.grid() # stile di linea predefinito
axis.grid(linestyle='-)') # linee della griglia solide
```

```
# -----
def _destroyWindow():
    root.quit()
    root.destroy()
# -----
radice = tk.Tk()
root.ritira()
root.protocol('WM_DELETE_WINDOW', _destroyWindow)
# -----
canvas = FigureCanvasTkAgg(fig, master=root)
canvas._tkcanvas.pack(side=tk.TOP, fill=tk.BOTH, expand=1)
# -----
root.update()
root.deiconify()
root.mainloop()
```

L'esecuzione del codice precedente produce il seguente grafico:



Nella prima riga di codice, dopo le istruzioni import, creiamo un'istanza di a figura oggetto. Successivamente, aggiungiamo sottotrame a questa figura chiamando add_subplot(211). Il primo numero in 211 indica alla figura quanti grafici aggiungere, il secondo numero determina il numero di colonne e il terzo indica alla figura l'ordine in cui visualizzare i grafici.

Questo potrebbe diventare più chiaro fornendo un esempio basato sul sito Web Matplotlib:

```
da matplotlib.figure import Figura
da matplotlib.backends.backend_tkagg import FigureCanvasTkAgg import tkinter as
tk
# -----
fig = Figure(figsize=(12, 8), facecolor='bianco')
```

```
xValori = [1,2,3,4]
yValues  = [5,7,6,8]
# -----
asse1 = fig.add_subplot(221)
asse2 = fig.add_subplot(222, sharex=axis1, sharey=axis1) axis3 =
fig.add_subplot(223, sharex=axis1, sharey=axis1) axis4 =
fig.add_subplot(224, sharex=axis1, sharey=axis1)
# -----
axis1.plot(xValues, yValues) axis1.set_xlabel('Horizontal
Label 1') axis1.set_ylabel('Vertical Label 1')
axis1.grid(linestyle='-) # linee griglia solide

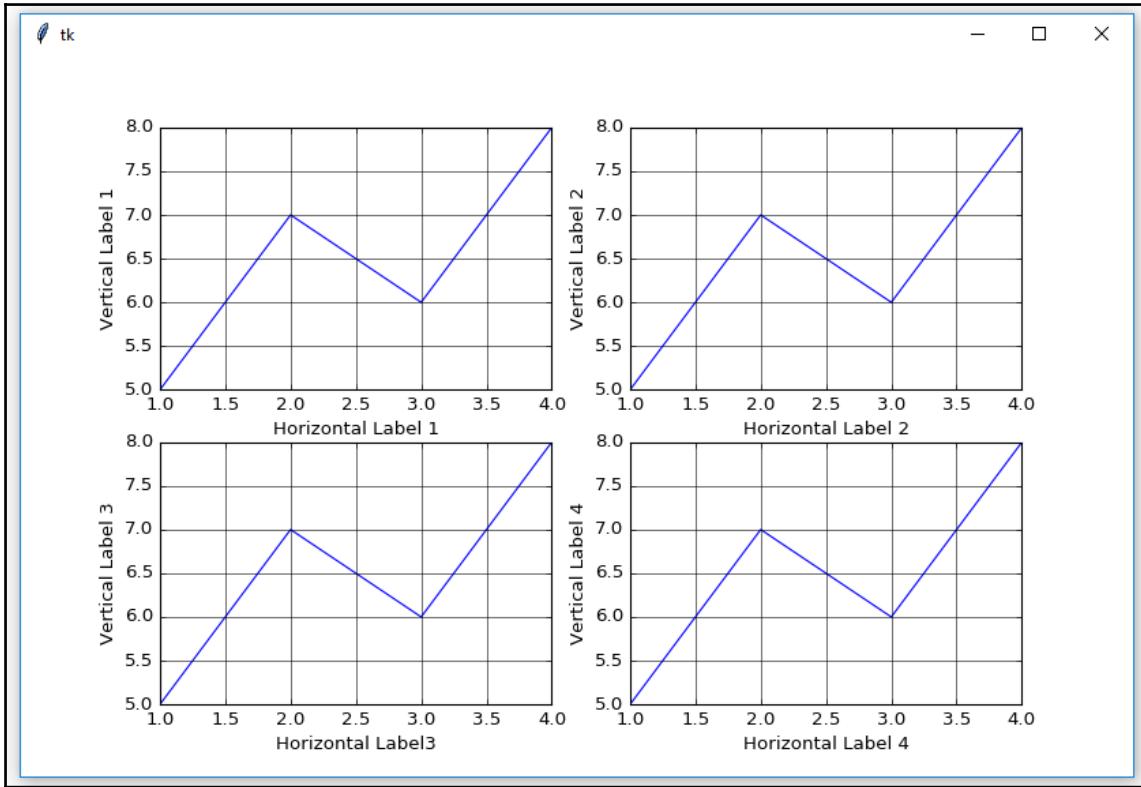
# -----
axis2.plot(xValues, yValues) axis2.set_xlabel('Horizontal
Label 2') axis2.set_ylabel('Vertical Label 2')
axis2.grid(linestyle='-) # linee griglia solide

# -----
axis3.plot(xValues, yValues) axis3.set_xlabel('Horizontal
Label3') axis3.set_ylabel('Vertical Label 3')
axis3.grid(linestyle='-) # linee griglia solide

# -----
axis4.plot(xValues, yValues) axis4.set_xlabel('Horizontal
Label 4') axis4.set_ylabel('Vertical Label 4')
axis4.grid(linestyle='-) # linee griglia solide

# -----
def _destroyWindow():
    root.quit()
    root.destroy()
# -----
radice = tk.Tk()
root.ritira()
root.protocol('WM_DELETE_WINDOW', _destroyWindow)
# -----
canvas = FigureCanvasTkAgg(fig, master=root)
canvas._tkcanvas.pack(side=tk.TOP, fill=tk.BOTH, expand=1)
# -----
root.update()
root.deiconify()
root.mainloop()
```

L'esecuzione del codice precedente comporta la creazione del seguente grafico:



La cosa importante da notare qui è che creiamo un asse, che viene quindi utilizzato come condiviso *X* e *y* assi per gli altri grafici all'interno del grafico. In questo modo, possiamo ottenere un layout del grafico simile a un database.

Aggiungiamo anche una griglia e cambiamo il suo stile di linea predefinito. Anche se mostriamo solo un grafico nel grafico, scegliendo 2 per il numero di sottotrame, stiamo spostando la trama verso l'alto, il che si traduce in uno spazio bianco in più nella parte inferiore del grafico. Questo primo grafico ora occupa solo il 50% dello schermo, il che influisce sulla dimensione delle linee della griglia di questo grafico quando vengono visualizzate.



Sperimenta con il codice decommentando il codice per asse = e
asse.griglia() per vedere i diversi effetti.

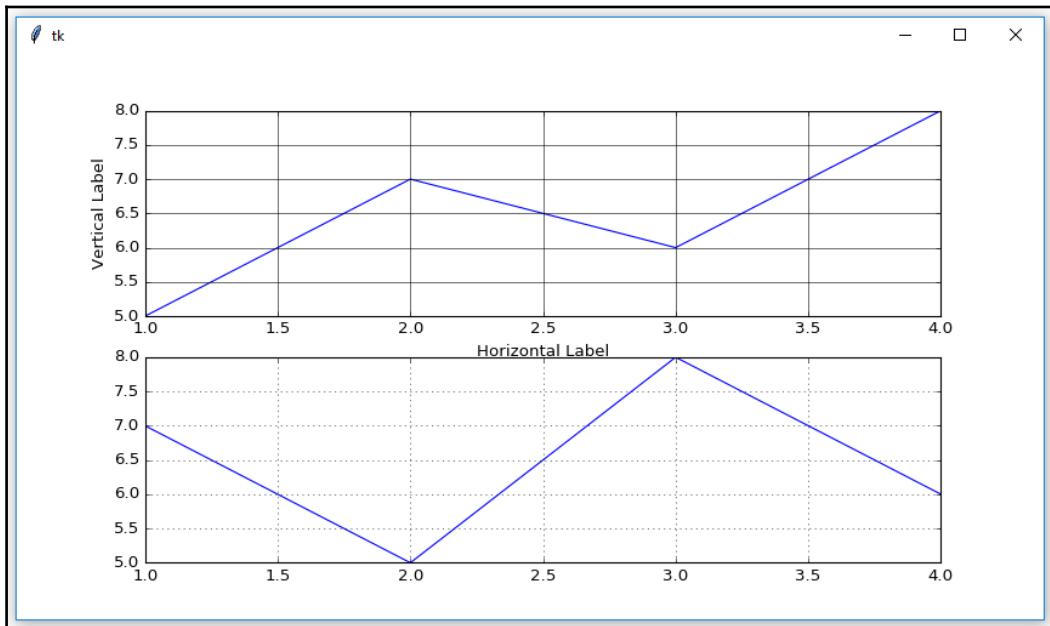
Possiamo aggiungere più sottotrame assegnandole alla seconda posizione usando add_subplot(212):

Matplotlib_labels_two_charts.py

```
da matplotlib.figure import Figura
da matplotlib.backends.backend_tkagg import FigureCanvasTkAgg import tkinter as
tk
# -----
fig = Figure(figsize=(12, 8), facecolor='bianco')
# -----
asse = fig.add_subplot(211) # 2 righe, 1 colonna, grafico in alto
# -----
xValori = [1,2,3,4]
yValues  = [5,7,6,8]
axis.plot(xValues, yValues) axis.set_xlabel('Horizontal Label')
axis.set_ylabel('Vertical Label') axis.grid(linestyle='-') # linee griglia
solide

# -----
asse1 = fig.add_subplot(212) # 2 righe, 1 colonna, grafico in basso
# -----
xValues1 = [1,2,3,4]
yValues1 = [7,5,8,6]
axis1.plot(xValues1, yValues1)
axis1.grid() # stile di linea predefinito
# -----
def _destroyWindow():
    root.quit()
    root.destroy()
# -----
radice = tk.Tk()
root.ritira()
root.protocol('WM_DELETE_WINDOW', _destroyWindow)
# -----
canvas = FigureCanvasTkAgg(fig, master=root)
canvas._tkcanvas.pack(side=tk.TOP, fill=tk.BOTH, expand=1)
# -----
root.update()
root.deiconify()
root.mainloop()
```

L'esecuzione del codice leggermente modificato ora aggiunge asse1 al grafico. Per la griglia del grafico in basso, abbiamo lasciato lo stile della linea come predefinito:



Come funziona...

Abbiamo importato il necessario Matplotlib moduli per creare una figura e una tela su cui disegnare il grafico. Gli abbiamo dato alcuni valori per il *X*e *y* assi e impostare alcune delle numerose opzioni di configurazione.

Abbiamo creato il nostro tkinter finestra in cui visualizzare il grafico, e abbiamo personalizzato il posizionamento dei grafici.

Come abbiamo visto nei capitoli precedenti, per creare un create tkinter GUI, dobbiamo prima importare il tkinter modulo e quindi creare un'istanza di Tk classe. Assegniamo questa istanza di classe a una variabile che chiamiamoradice, che è un nome spesso usato negli esempi che trovi online.

Nostro tkinter La GUI non diventerà visibile finché non avvieremo il ciclo dell'evento principale e, per farlo, noi usiamo root.mainloop().

Come dare al grafico una legenda

Una volta che iniziamo a tracciare più di una linea di punti dati, le cose potrebbero diventare un po' poco chiare. Quindi, aggiungendo una legenda ai nostri grafici, possiamo dire quali dati sono cosa e cosa significano effettivamente.

Non dobbiamo scegliere colori diversi per rappresentare i diversi dati. Matplotlib assegna automaticamente un colore diverso a ciascuna linea dei punti dati.

Tutto quello che dobbiamo fare è creare il grafico e aggiungervi una legenda.

Prepararsi

In questa ricetta andremo ad arricchire lo schema della ricetta precedente, *Posizionamento di etichette sui grafici*. Tracciamo solo un grafico.

Come farlo...

Innanzitutto, tracceremo più righe di dati sullo stesso grafico, quindi aggiungeremo una legenda al grafico.

Lo faremo modificando il codice della ricetta precedente:

Matplotlib_chart_with_legend.py

```
da matplotlib.figure import Figura
da matplotlib.backends.backend_tkagg import FigureCanvasTkAgg import tkinter as
tk
# -----
fig = Figure(figsize=(12, 5), facecolor='bianco')
# -----
asse = fig.add_subplot(111) # 1 riga, 1 colonna

xValori = [1,2,3,4]

yValues0 = [6,7,5,8,7,5] yValues1
= [5,5,6,5,8,6] yValues2 =
[6,5,7,8,7]

t0, = axis.plot(xValues, yValues0) t1, =
axis.plot(xValues, yValues1) t2, =
axis.plot(xValues, yValues2)
```

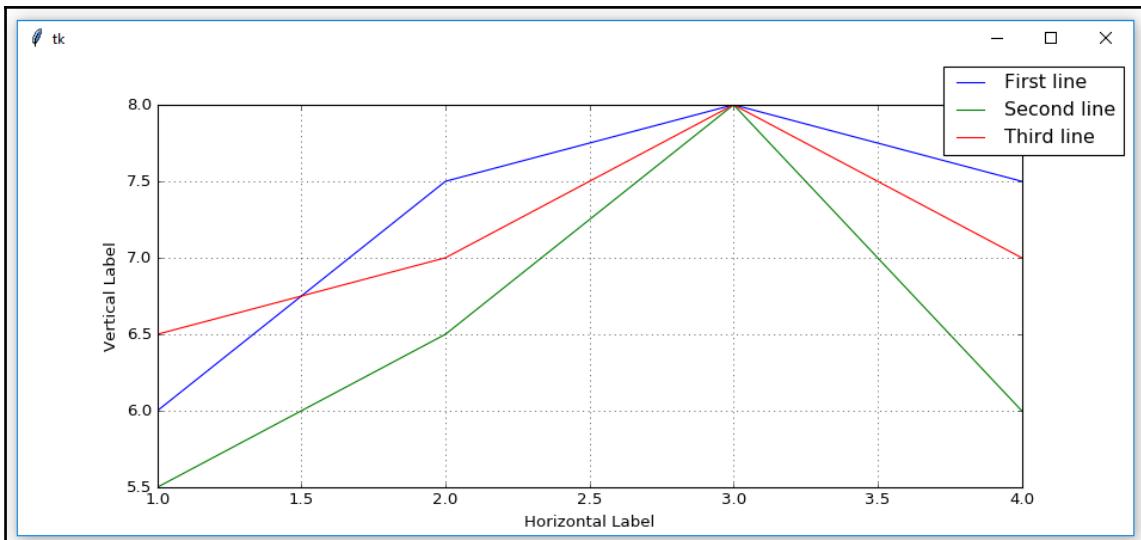
```
axis.set_ylabel('Etichetta Verticale')
axis.set_xlabel('Etichetta Orizzontale')

asse.griglia()

fig.legend((t0, t1, t2), ('Prima riga', 'Seconda riga', 'Terza
riga'), 'in alto a destra')

# -----
def _destroyWindow():
    root.quit()
    root.destroy()
# -----
radice = tk.Tk()
root.ritira()
root.protocol('WM_DELETE_WINDOW', _destroyWindow)
# -----
canvas = FigureCanvasTkAgg(fig, master=root)
canvas._tkcanvas.pack(side=tk.TOP, fill=tk.BOTH, expand=1)
# -----
root.update()
root.deiconify()
root.mainloop()
```

L'esecuzione del codice modificato crea il seguente grafico, che ha una legenda nell'angolo in alto a destra:



Stiamo tracciando solo un grafico in questa ricetta e lo facciamo modificando `fig.add_subplot(111)`. Modifichiamo leggermente anche la dimensione della figura tramite il `figsize` proprietà.

Successivamente, creiamo tre elenchi Python che contengono i valori da tracciare. Quando tracciamo i dati, salviamo i riferimenti ai grafici nelle variabili locali.

Creiamo la legenda passando in una tupla con i riferimenti ai tre grafici, un'altra tupla che contiene le stringhe che vengono poi visualizzate nella legenda, e nel terzo argomento posizioniamo la legenda all'interno del grafico.

Le impostazioni predefinite di Matplotlib assegnare uno schema di colori alle linee da tracciare.

Possiamo facilmente modificare questa impostazione predefinita dei colori con i colori che preferiamo impostando una proprietà quando tracciamo ciascun asse.

Lo facciamo usando il colore proprietà e assegnandole un valore di colore disponibile:

`Matplotlib_chart_with_legend.py`

```
t0, = axis.plot(xValues, yValues0, color = 'viola') t1, = axis.plot(xValues,  
yValues1, color = 'red') t2, = axis.plot(xValues, yValues2, color = 'blu' )
```

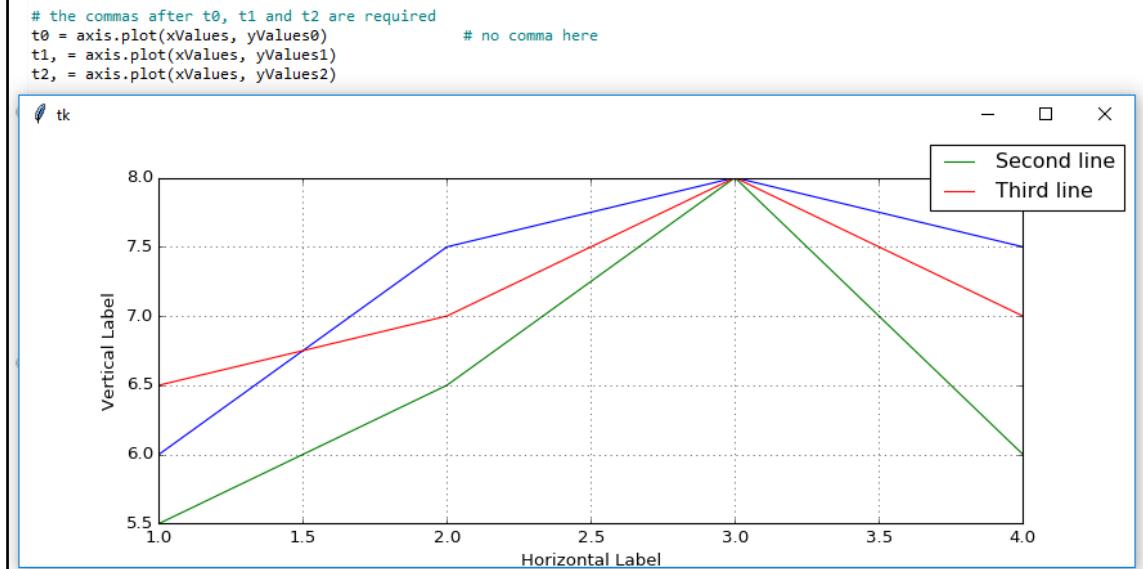
Nota che la virgola dopo le assegnazioni delle variabili di `t0`, `t1`, e `t2` non è un errore. È necessario per creare la legenda.

La virgola dopo ogni variabile trasforma una lista in una tupla. Se lo tralasciamo, la nostra legenda non verrà visualizzata.

Il codice verrà comunque eseguito, solo senza la legenda prevista.



Quando rimuoviamo la virgola dopo il `t0` assegnazione, otteniamo un errore e la prima riga non compare più nella figura. Il grafico e la legenda vengono comunque creati ma senza che la prima riga venga visualizzata nella legenda.



Come funziona...

Abbiamo migliorato il nostro grafico tracciando tre linee di dati nello stesso grafico e assegnandogli una legenda per distinguere i dati tracciati da quelle tre linee.

Grafici di ridimensionamento

Nelle ricette precedenti, mentre creavamo i nostri primi grafici e li miglioravamo, abbiamo codificato il ridimensionamento di come quei valori sono rappresentati visivamente.

Anche se questo ci è servito bene per i valori che stavamo usando, spesso tracciamo grafici da database molto grandi.

A seconda dell'intervallo di tali dati, i nostri valori hardcoded per il verticale y -dimensione potrebbe non essere sempre la soluzione migliore e potrebbe rendere difficile vedere le linee nei nostri grafici.

Prepararsi

Miglioreremo il nostro codice dalla ricetta precedente, *Come dare al grafico una legenda*. Se non hai digitato tutto il codice delle ricette precedenti, scarica semplicemente il codice per questo capitolo e inizierà (e poi potrai divertirti molto a creare GUI, grafici e così via, usando Python).

Come farlo...

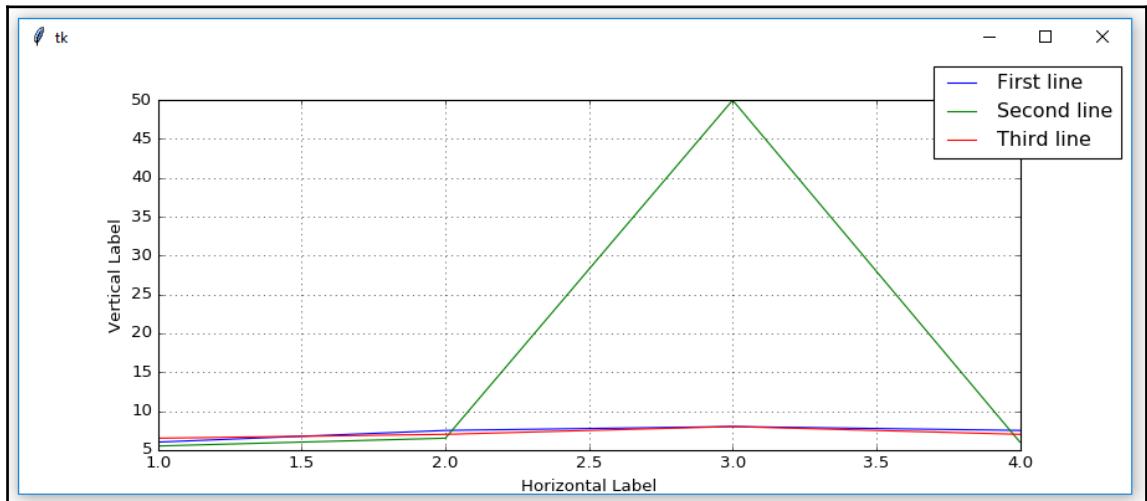
Modifica il yValues1 riga di codice della ricetta precedente da usare 50 come terzo valore:

Matplotlib_labels_two_charts_not_scaled.py

```
asse = fig.add_subplot(111) xValues           # 1 riga, 1 colonna
= [1,2,3,4]
yValues0 = [6,7.5,8,7.5] yValues1
= [5.5,6.5,50,6] yValues2 =                 # un valore molto alto (50)
[6.5,7,8,7]
```

L'unica differenza rispetto al codice che ha creato il grafico nella ricetta precedente è un valore di dati.

Modificando un valore che non è vicino all'intervallo medio di tutti gli altri valori per tutte le linee tracciate, la rappresentazione visiva dei dati è cambiata drasticamente, abbiamo perso molti dettagli sui dati complessivi e ora vediamo principalmente un picco elevato :



Finora, i nostri grafici si sono adattati in base ai dati che rappresentano visivamente.

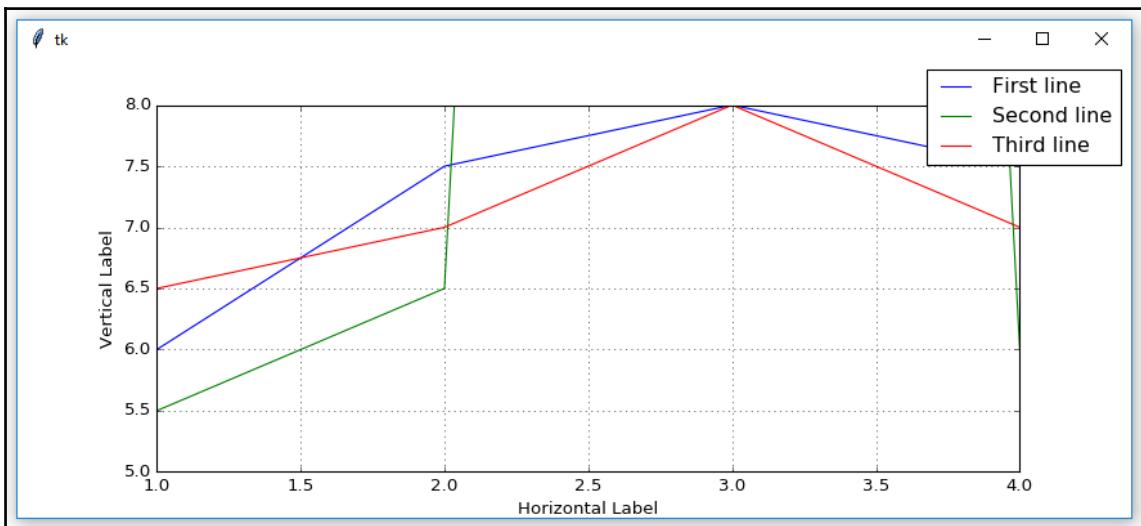
Anche se questa è una caratteristica pratica di Matplotlib, questo non è sempre quello che vogliamo. Possiamo restringere la scala del grafico rappresentato limitando la verticale y-dimensione:

Matplotlib_labels_two_charts_scaled.py

```
yValues0 = [6,7.5,8,7.5] yValues1  
= [5.5,6.5,50,6] yValues2 = # un valore molto alto (50)  
[6.5,7,8,7]  
  
asse.set_ylim(5, 8) # limita la visualizzazione verticale
```

Il `asse.set_ylim(5, 8)` la riga di codice ora limita il valore iniziale a 5 e il valore finale del display verticale a 8.

Ora, quando creiamo il nostro grafico, il picco di valore elevato non ha più l'impatto che aveva prima:



Come funziona...

Abbiamo aumentato un valore nei dati, con un effetto drammatico. Impostando dei limiti alla visualizzazione verticale e orizzontale del grafico, possiamo vedere i dati che ci interessano di più.

Anche le punte, come quelle appena mostrate, possono essere di grande interesse. Tutto dipende da cosa stiamo cercando. La rappresentazione visiva dei dati è di grande valore.



Un'immagine vale più di mille parole.

Regolazione dinamica della scala dei grafici

Nella ricetta precedente, abbiamo imparato come possiamo limitare il ridimensionamento dei nostri grafici. In questa ricetta, faremo un ulteriore passo avanti regolando dinamicamente il ridimensionamento impostando sia un limite che analizzando i nostri dati prima di rappresentarli.

Prepararsi

Miglioreremo il codice della ricetta precedente, *Grafici di ridimensionamento*, leggendo i dati stiamo tracciando dinamicamente, calcolando la media e quindi aggiustando il nostro grafico.

Anche se in genere leggiamo i dati da una fonte esterna, in questa ricetta creeremo i dati che stiamo tracciando usando gli elenchi Python, come si può vedere nel codice nella sezione seguente.

Come farlo...

Stiamo creando i nostri dati nel nostro modulo Python assegnando liste con dati ai xValues e yValues variabili.

In molti grafici, l'inizio del *X* e sì il sistema di coordinate inizia da (0, 0). Questa di solito è una buona idea, quindi regoliamo di conseguenza il nostro codice delle coordinate del grafico.

Modifichiamo il codice per impostare dei limiti su entrambi i *X* e *y* dimensioni:

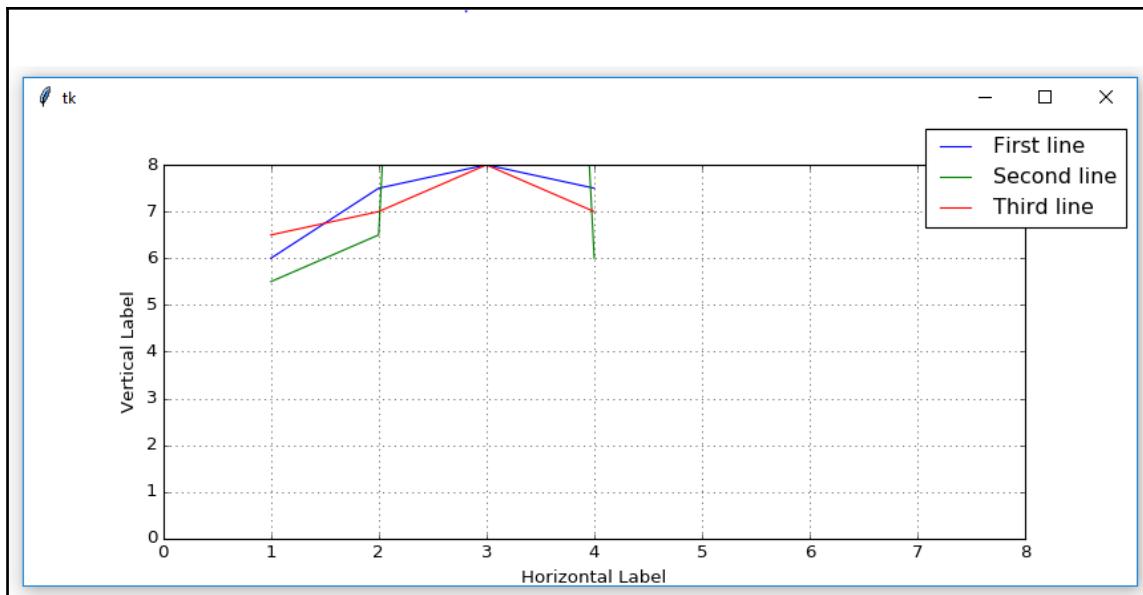
Matplotlib_labels_two_charts_scaled_dynamic.py

```
xValori = [1,2,3,4]

yValues0 = [6,7.5,8,7.5] yValues1
= [5.5,6.5,50,6] yValues2 = # un valore molto alto (50)
[6.5,7,8,7]

asse.set_ylim(0, 8) # limite inferiore (0)
asse.set_xlim(0, 8) # usa gli stessi limiti per x
```

Ora che abbiamo fissato gli stessi limiti per *X* e *y*, il nostro grafico potrebbe sembrare più equilibrato.
Quando eseguiamo il codice modificato, otteniamo il seguente risultato:



Forse iniziare da (0, 0) non è stata una grande idea dopotutto...

Quello che vogliamo veramente fare è regolare il nostro grafico in modo dinamico in base all'intervallo dei dati, limitando allo stesso tempo i valori che sono troppo alti o troppo bassi.

Possiamo farlo analizzando tutti i dati da rappresentare nel grafico e, allo stesso tempo, ponendo dei limiti esplicativi.

Modificare il codice come segue:

Matplotlib_labels_two_charts_scaled_dynamic.py

```
xValori = [1,2,3,4]

yValues0 = [6,7.5,8,7.5] yValues1
= [5.5,6.5,50,6] yValues2 = # un valore molto alto (50)
[6.5,7,8,7]
yAll = [yValues0, yValues1, yValues2] # elenco di elenchi

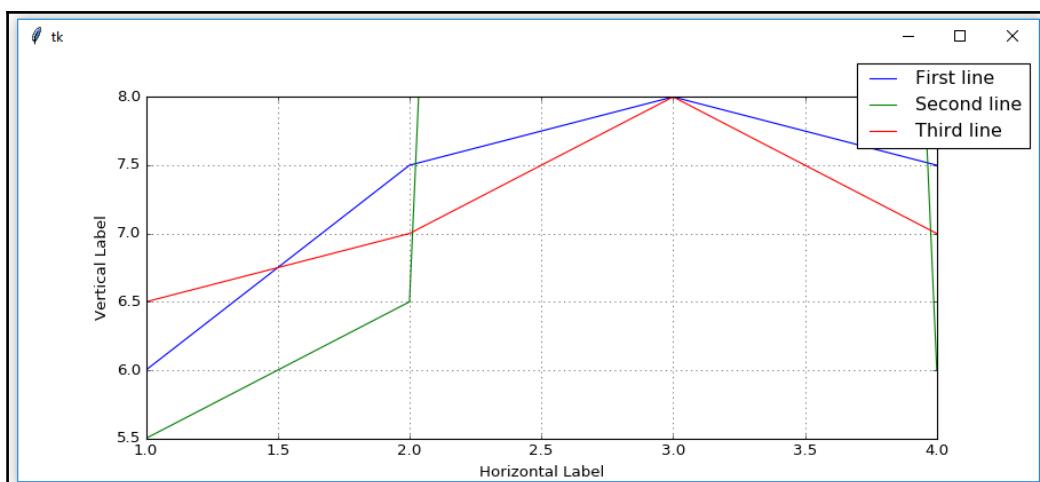
# appiattisci l'elenco delle liste che recuperano il valore minimo
minY = min([y per yValues in yAll for y in yValues])

yUpperLimit = 20
# appiattisci l'elenco degli elenchi recuperando il valore massimo entro il limite definito
maxY = max([y per yValues in yAll for y in yValues se y <
yUpperLimit])

# limiti dinamici
axis.set_xlim(min(xValues),
max(xValues))

t0, = axis.plot(xValues, yValues0) t1, =
axis.plot(xValues, yValues1) t2, =
axis.plot(xValues, yValues2)
```

L'esecuzione del codice risulta nel grafico seguente. Abbiamo regolato entrambi i suoi *X* e *y* dimensioni in modo dinamico. Nota come la dimensione ora inizia da **5,5** invece di **5,0**, come faceva prima. Inoltre, il grafico non inizia più da (0, 0), fornendoci informazioni più preziose sui nostri dati:



Stiamo creando una lista di liste per il *y*-dati dimensionali e quindi utilizzando una comprensione di elenco racchiusa in una chiamata a Python's min() e massimo() funzioni.

Se la comprensione delle liste sembra essere un po' avanzata, in pratica sono un ciclo molto compresso.

Sono inoltre progettati per essere più veloci di un normale ciclo di programmazione.

Nel codice Python precedente, abbiamo creato tre elenchi che contengono il *y*-dati dimensionali da tracciare.

Abbiamo quindi creato un altro elenco che contiene questi tre elenchi, che crea un elenco di elenchi, come segue:

```
yValues0 = [6,7.5,8,7.5] yValues1  
= [5.5,6.5,50,6] yValues2 = # un valore molto alto (50)  
[6.5,7,8,7]  
yAll = [yValues0, yValues1, yValues2] # elenco di elenchi
```

Siamo interessati ad ottenere sia il valore minimo di tutti i *y*-dati dimensionali e il valore massimo contenuto all'interno di questi tre elenchi.

Possiamo farlo tramite una comprensione dell'elenco Python:

```
# appiattisci l'elenco delle liste che recuperano il valore minimo  
minY = min([y for y in yAll for y in yValues])
```

Dopo aver eseguito la comprensione dell'elenco, minY è 5.5.

La riga di codice precedente è la comprensione delle liste che percorre tutti i valori di tutti i dati contenuti nelle tre liste e trova il valore minimo usando Python min parola chiave.

Nello stesso modello, troviamo il valore massimo contenuto nei dati che vogliamo tracciare. Questa volta, imposteremo anche un limite all'interno della nostra comprensione dell'elenco, che ignora tutti i valori che sono al di sopra del limite che abbiamo specificato, come segue:

```
yUpperLimit = 20  
# appiattisci l'elenco degli elenchi recuperando il valore massimo entro il limite definito  
maxY = max([y for y in yAll for y in yValues if y < yUpperLimit])
```

Dopo aver eseguito il codice precedente con la nostra restrizione scelta, maxY ha il valore di 8 (non 50).

Abbiamo applicato una restrizione al valore massimo, secondo una condizione predefinita scegliendo 20 come valore massimo da visualizzare nel grafico.

Per il *X*-dimensione, abbiamo semplicemente chiamato min() e massimo() nel Matplotlib metodo per ridimensionare dinamicamente i limiti del grafico.

Come funziona...

In questa ricetta ne abbiamo create diverse Matplotlib grafici e aggiustato alcune delle molte proprietà disponibili. Abbiamo anche usato il core Python per controllare dinamicamente il ridimensionamento dei grafici.

6

Thread e networking

In questo capitolo creeremo thread, code e socket TCP/IP utilizzando Python 3.6 e versioni successive. Tratteremo le seguenti ricette:

- Come creare più thread Avviare
- un thread
- Interrompere un thread
- Come usare le code
- Passaggio di code tra moduli diversi
- Utilizzo di widget di dialogo per copiare file sulla rete
- Utilizzo di TCP/IP per comunicare tramite reti Utilizzo di
- urlopen leggere i dati dai siti web

introduzione

In questo capitolo, estenderemo le funzionalità della nostra GUI Python utilizzando thread, code e connessioni di rete.



UN tkinter La GUI è un'applicazione a thread singolo. Ogni funzione che coinvolge il tempo di attesa o di sospensione deve essere chiamata in un thread separato; altrimenti, iltkinter La GUI si blocca.

Quando eseguiamo la nostra GUI Python in Task Manager di Windows, possiamo vedere che un nuovo python.exe è stato avviato il processo. Quando diamo alla nostra GUI Python un file .pyw estensione, allora il processo creato sarà pitone.pyw, come si può vedere nel Task Manager.

Quando viene creato un processo, il processo crea automaticamente un thread principale per eseguire la nostra applicazione. Questa è chiamata applicazione a thread singolo.

Per la nostra GUI Python, un'applicazione a thread singolo porterà la nostra GUI a bloccarsi non appena chiamiamo un'attività in esecuzione più lunga, come fare clic su un pulsante che ha un tempo di sospensione di alcuni secondi. Per mantenere la nostra GUI reattiva, dobbiamo usare il multithreading, e questo è ciò che studieremo in questo capitolo. Possiamo anche creare più processi creando più istanze della nostra GUI Python, come si può vedere nel Task Manager.

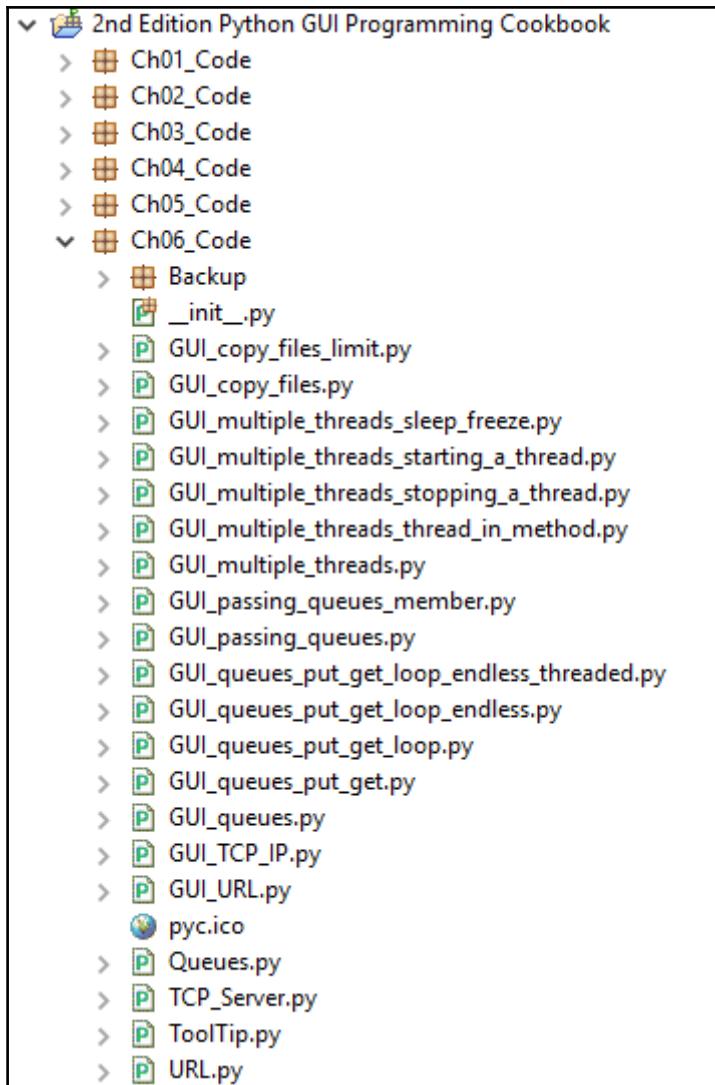
I processi sono isolati l'uno dall'altro in base alla progettazione e non condividono dati comuni. Per comunicare tra processi separati, dovremmo usare**Comunicazione tra processi (IPC)**, che è una tecnica avanzata. I thread, d'altra parte, condividono dati, codice e file comuni, il che rende la comunicazione tra i thread all'interno dello stesso processo molto più semplice rispetto all'utilizzo di IPC.



Un'ottima spiegazione dei thread può essere trovata su https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/4_Threads.html.

In questo capitolo impareremo come mantenere la nostra GUI Python reattiva e impedire che si blocchi.

Ecco la panoramica dei moduli Python per questo capitolo:



Come creare più thread

Creeremo più thread usando Python. Ciò è necessario per mantenere la nostra GUI reattiva.



Un filo è come tessere un tessuto fatto di filato e non c'è nulla di cui aver paura.

Prepararsi

Più thread vengono eseguiti nello stesso spazio di memoria del processo del computer. Non c'è bisogno di IPC, che complicherebbe il nostro codice. In questa ricetta, eviteremo l'IPC utilizzando i thread.

Come farlo...

Per prima cosa aumenteremo le dimensioni del nostro Testo scorrevole widget, rendendolo più grande. aumentiamo scroll_w per 40 e scroll_h per 10:

```
# Utilizzo di un controllo di testo a scorrimento
scroll_w = 40; scroll_h = 10                                     # aumenta le taglie
self.scrol = scrolledtext.ScrolledText(mighty, width=scroll_w,
                                         altezza=scroll_h,
                                         wrap=tk.WORD)
self.scrol.grid(column=0, row=3, sticky='WE', columnspan=3)
```

Quando ora eseguiamo la GUI risultante, il Spinbox widget è allineato al centro rispetto al widget Entry sopra di esso, il che non ha un bell'aspetto. Lo cambieremo allineando a sinistra il widget.

Inserisci appiccicoso='W' al griglia controllo per allineare a sinistra il Spinbox aggeggio:

```
# Aggiunta di un widget Casella rotante
self.spin = Spinbox(mighty, values=(1, 2, 4, 42, 100), width=5,
                     bd=9, comando=self._spin)
self.spin.grid(colonna=0, riga=2, sticky='W')                      # allineare a sinistra
```

La GUI potrebbe ancora avere un aspetto migliore, quindi ora aumenteremo le dimensioni del widget Entry per ottenere un layout GUI più equilibrato.

Aumentare il larghezza per 24, come mostrato nel seguente frammento di codice:

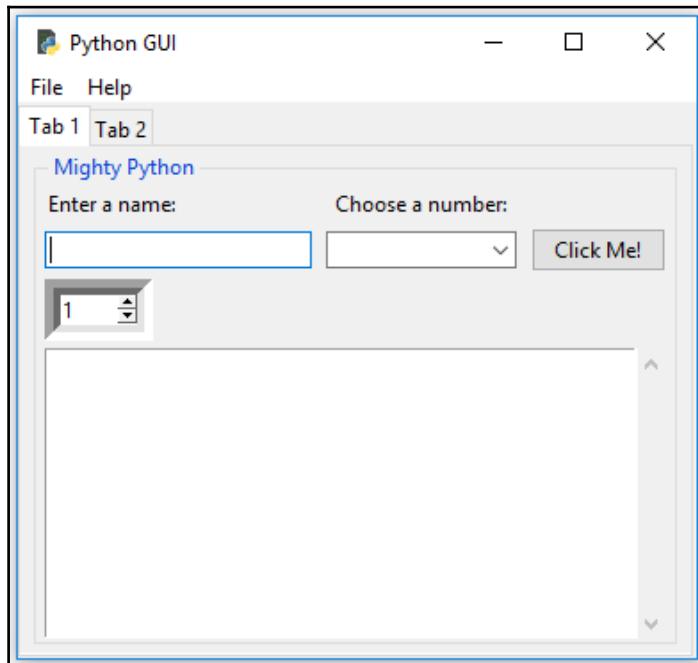
```
# Aggiunta di un widget Voce casella di testo
self.name = tk.StringVar()
self.name_entered = ttk.Entry (potente, larghezza = 24,
                               textvariable=self.name)
self.name_entered.grid(colonna=0, riga=1, appiccicoso='W')
```

Aumentiamo anche leggermente la larghezza di Casella combinata per 14:

```
ttk.Label(mighty, text="Scegli un numero:").grid(column=1, row=0) numero =  
tk.StringVar()  
self.number_chosen = ttk.Combobox(potente, larghezza=14,  
                                   variabile di testo=numero,  
                                   stato='sola lettura')  
self.number_chosen['values'] = (1, 2, 4, 42, 100)  
self.number_chosen.grid(colonna=1, riga=1)  
self.number_chosen.current(0)
```

L'esecuzione del codice modificato e migliorato si traduce in una GUI più grande, che utilizzeremo per questa e le seguenti ricette:

GUI_multiple_threads.py



Per creare e utilizzare i thread in Python, dobbiamo importare il file Filo classe dal filettaura modulo:

```
# ======  
# importazioni  
# ======  
importa tkinter come tk
```

```
da tkinter import ttk
from tkinter import scrolledtext
from tkinter import Menu
from tkinter import messagebox as msg
from tkinter import Spinbox
from time import sleep
from Ch06_Code.ToolTip import tt
import threading
import Thread

GLOBAL_CONST = 42
```

Aggiungiamo un metodo da creare in un thread alla nostra classe OOP:

```
classe OOP():
    def metodo_in_a_thread(self):
        print('Ciao, come stai?')
```

Ora possiamo chiamare il nostro metodo threaded nel codice, salvando l'istanza in una variabile:

```
# =====
# Avvia GUI
# =====
oop = OOP()

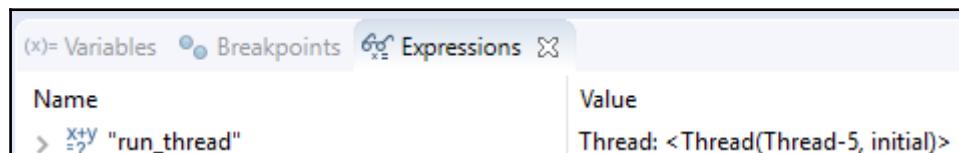
# Metodi in esecuzione nei thread
run_thread = Thread(target=oop.method_in_a_thread)

oop.win.mainloop()
```

Ora abbiamo un metodo con thread, ma quando eseguiamo il codice, non viene stampato nulla sulla console!

Dobbiamo iniziare il thread prima che possa essere eseguito e la prossima ricetta ci mostrerà come farlo.

Tuttavia, l'impostazione di un punto di interruzione dopo il ciclo dell'evento principale della GUI dimostra che abbiamo effettivamente creato un oggetto thread, come si può vedere nel debugger IDE di Eclipse:



Come funziona...

In questa ricetta, abbiamo preparato la nostra GUI per utilizzare i thread aumentando prima la dimensione della GUI in modo da poter vedere meglio i risultati stampati sul Testo scorrevole aggeggio.

Abbiamo quindi importato il Filo classe dal Python filettatura modulo. Successivamente abbiamo creato un metodo che chiamiamo in un thread all'interno della nostra GUI.

Avvio di un thread

Questa ricetta ci mostrerà come iniziare un thread. Dimostrerà anche perché i thread sono necessari per mantenere la nostra GUI reattiva durante le attività di lunga durata.

Prepararsi

Vediamo prima cosa succede quando chiamiamo una funzione o un metodo della nostra GUI che ne ha dormire associato ad esso senza utilizzare thread.

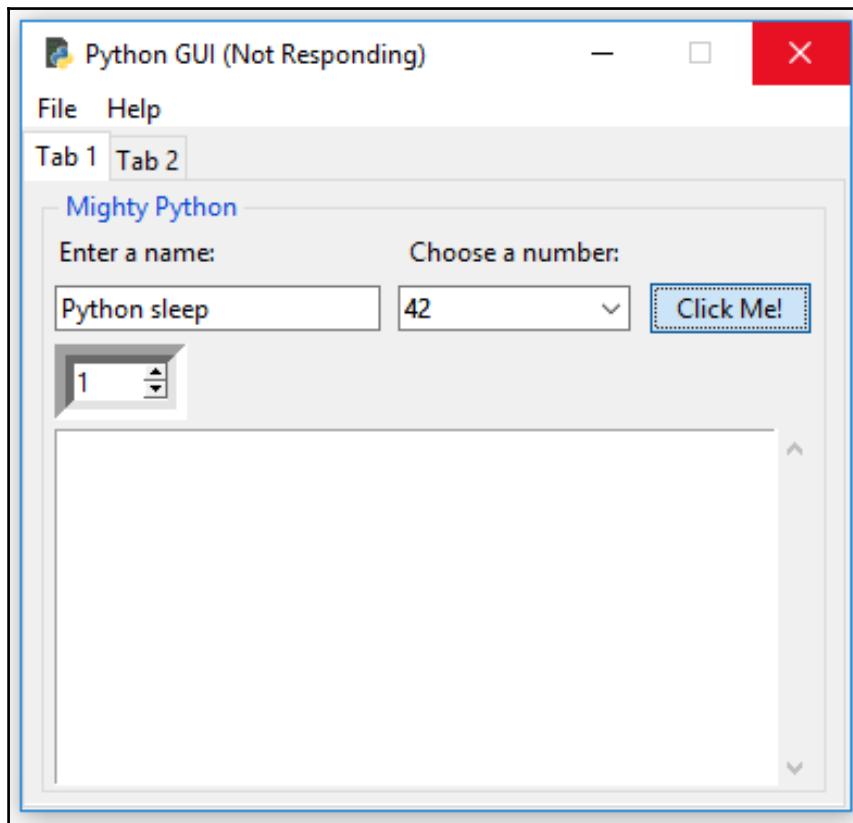


stiamo usando dormire qui per simulare un'applicazione del mondo reale che potrebbe dover attendere la risposta di un server Web o di un database, un trasferimento di file di grandi dimensioni o calcoli complessi per completare il proprio compito.
dormire è un segnaposto molto realistico e mostra il principio coinvolto.

L'aggiunta di un ciclo nel nostro metodo di callback del pulsante con un po' di tempo di sospensione fa sì che la nostra GUI non risponda e, quando proviamo a chiudere la GUI, le cose peggiorano ancora:
`GUI_multiple_threads_sleep_freeze.py`

```
# Pulsante di richiamata
def click_me(self):
    self.action.configure(text='Hello ' + self.name.get() + ''
                          + self.number_chosen.get())
    # Il codice senza thread con sleep blocca la GUI
    per idx nell'intervallo (10):
        dormire(5)
        self.scrol.insert(tk.INSERT, str(idx) + 'n')
```

L'esecuzione del file di codice precedente risulta nella seguente schermata:



Se aspettiamo abbastanza a lungo, il metodo alla fine verrà completato, ma durante questo periodo nessuno dei nostri widget della GUI risponde agli eventi di clic. Risolviamo questo problema utilizzando i thread.



Nella ricetta precedente, abbiamo creato un metodo da eseguire in un thread ma, finora, il thread non è stato eseguito!

A differenza delle normali funzioni e metodi Python, dobbiamo `we` inizio un metodo che verrà eseguito nel proprio thread!

Questo è quello che faremo dopo.

Come farlo...

Per prima cosa, spostiamo la creazione del thread nel suo metodo e poi chiamiamo questo metodo dal pulsante richiamato:

```
# Metodi in esecuzione nei thread
def create_thread(self):
    self.run_thread = Thread(target=self.method_in_a_thread)
    self.run_thread.start() # avvia il thread

# Pulsante di richiamata
def click_me(self):
    self.action.configure(text='Hello ' + self.name.get())
    self.create_thread()
```

Facendo clic sul pulsante ora si ottiene il `create_thread` metodo chiamato, che a sua volta chiama il `method_in_a_thread` metodo.

Innanzitutto, creiamo un thread e lo indirizziamo a un metodo. Successivamente, avviamo il thread che esegue il metodo mirato in un nuovo thread:

```
def method_in_a_thread(self):
    print('Hi, how are you?')

# Running methods in Threads
def create_thread(self):
    self.run_thread = Thread(target=self.method_in_a_thread)
    self.run_thread.start()

# Button callback
def click_me(self):
    self.action.configure(text='Hello ' + self.name.get())
    self.create_thread()
```

The screenshot shows the Eclipse IDE interface with a Python script open. The script contains code for creating a GUI with a text entry field, a dropdown menu, and a button. It also includes logic to run a method in a separate thread. Below the editor, a terminal window displays the output 'Hi, how are you?'.

Python GUI

File Help

Tab 1 Tab 2

Mighty Python

Enter a name: Choose a number:

Python

1

Hello Python

Search Console PyUnit

C:\Eclipse_NEON_workspace\2nd Edition Python GUI Programming Cookbook\Ch06_Code\GUI

Hi, how are you?



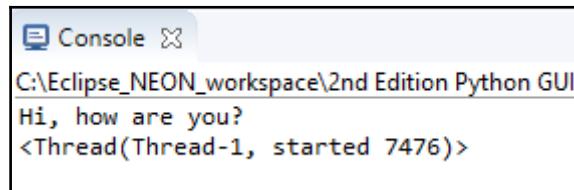
La GUI stessa viene eseguita nel proprio thread, che è il thread principale dell'applicazione.

Possiamo stampare l'istanza del thread:

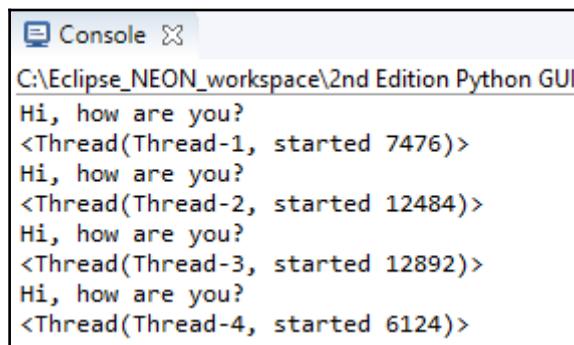
GUI_multiple_threads_thread_in_method.py

```
# Metodi in esecuzione nei thread
def create_thread(self):
    self.run_thread = Thread(target=self.method_in_a_thread)
    self.run_thread.start() # avvia il thread
    print(self.run_thread)
```

Facendo clic sul pulsante ora viene creata la seguente stampa:



Quando facciamo clic sul pulsante più volte, possiamo vedere che a ogni thread viene assegnato un nome e un ID univoci:

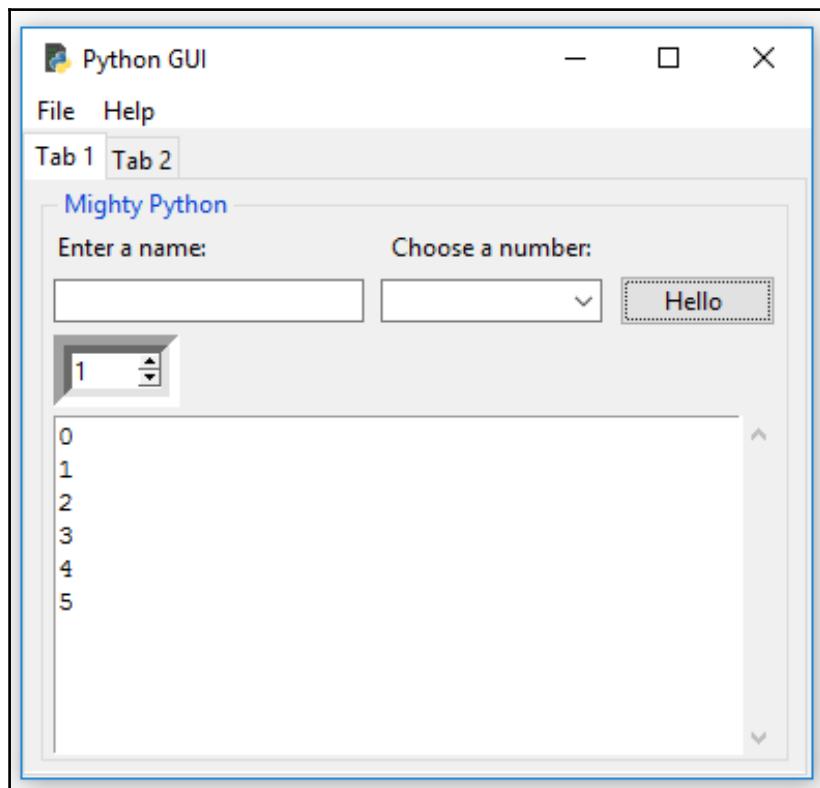


Ora spostiamo il nostro codice con dormire in un ciclo nel method_in_a_thread metodo per verificare che i thread risolvano davvero il nostro problema:

```
def metodo_in_a_thread(self):
    print('Ciao, come stai?') per idx in
    range(10):
        dormire(5)
        self.scrol.insert(tk.INSERT, str(idx) + 'n')
```

Quando si fa clic sul pulsante, mentre i numeri vengono stampati nel Testo scorrevole widget con un ritardo di cinque secondi, possiamo fare clic ovunque nella nostra GUI, cambiare scheda e così via. La nostra GUI è diventata di nuovo reattiva perché stiamo usando i thread!

GUI_multiple_threads_starting_a_thread.py



Come funziona...

In questa ricetta, abbiamo chiamato i metodi della nostra classe GUI nei propri thread e abbiamo appreso che dobbiamo avviare i thread. Altrimenti, il thread viene creato ma rimane lì in attesa che eseguiamo il suo metodo di destinazione.

Abbiamo notato che a ogni thread viene assegnato un nome e un ID univoci.

Abbiamo simulato attività di lunga durata inserendo a dormire istruzione nel nostro codice, che ci ha mostrato che i thread possono effettivamente risolvere il nostro problema.

Interrompere un thread

Dobbiamo iniziare un thread per farlo effettivamente fare qualcosa chiamando il `inizio()` metodo, così intuitivamente, ci aspetteremmo che ci sia una corrispondenza `fermare()` metodo, ma non esiste. In questa ricetta impareremo come eseguire un thread come attività in background, chiamata *ademon*. Quando si chiude il thread principale, che è la nostra GUI, anche tutti i demoni verranno automaticamente interrotti.

Prepararsi

Quando chiamiamo metodi in un thread, possiamo anche passare argomenti e argomenti di parole chiave al metodo. Iniziamo questa ricetta facendo esattamente questo.

Come farlo...

Aggiungendo `argomenti=[8]` al costruttore di `thread` e modificando il metodo mirato per aspettarsi argomenti, possiamo passare argomenti ai metodi con `thread`. Il parametro `perargomenti` deve essere una sequenza, quindi avvolgeremo il nostro numero in un elenco Python:

```
def method_in_a_thread(self, num_of_loops=10):
    print('Ciao, come stai?')
    per idx nell'intervallo (num_of_loops):
        dormire(5)
        self.scrol.insert(tk.INSERT, str(idx) + 'n')
```

Nel codice seguente, `run_thread` è una variabile locale, a cui accediamo solo nell'ambito del metodo all'interno del quale abbiamo creato `run_thread`:

```
# Metodi in esecuzione nei thread
def create_thread(self):
    run_thread = Thread(target=self.method_in_a_thread, args=[8])
    run_thread.start()
```

Trasformando la variabile locale in un membro, possiamo quindi verificare se il thread è ancora in esecuzione chiamando `è vivo` su di esso da un altro metodo:

```
# Metodi in esecuzione nei thread
def create_thread(self):
    self.run_thread = Thread(target=self.method_in_a_thread, args=
                             [8])
    self.run_thread.start()
    print(self.run_thread)
    print('createThread():', self.run_thread.isAlive())
```

Nel codice precedente, abbiamo elevato il nostro local `run_thread` variabile a un membro della nostra classe. Questo ci permette di accedere al `self.run_thread` variabile da qualsiasi metodo nella nostra classe.

Si ottiene così:

```
def method_in_a_thread(self, num_of_loops=10):
    per idx nell'intervallo (num_of_loops):
        dormire(1)
        self.scrol.insert(tk.INSERT, str(idx) + 'n') sleep(1)

    print('method_in_a_thread():', self.run_thread.isAlive())
```

Quando facciamo clic sul pulsante e quindi usciamo dalla GUI, possiamo vedere che le istruzioni di stampa nel `create_thread` sono stati stampati, ma non vediamo la seconda istruzione `print` da `method_in_a_thread`.

Considera il seguente output:

```
Console <terminated> C:\Eclipse_NEON_workspace\Hi, how are you?
<Thread(Thread-1, started 4800)>
createThread(): True
method_in_a_thread(): True
```

Invece dell'output precedente, otteniamo il seguente **Errore di runtime**:

The screenshot shows the Eclipse IDE interface with a code editor and a terminal window. The code editor contains Python code demonstrating threads. The terminal window (Console) shows the output of running this code, which includes a message from the GUI, the creation of a thread, and an exception indicating that the main thread is not in the main loop.

```
def method_in_a_thread(self, num_of_loops=10):
    print('Hi, how are you?')
    for idx in range(num_of_loops):
        sleep(1)
        self.scrol.insert(tk.INSERT, str(idx) + '\n')
    print('method_in_a_thread():', self.run_thread.isAlive())

# Running methods in Threads
def create_thread(self):
    self.run_thread = Thread(target=self.method_in_a_thread, args=[8])
    self.run_thread.start()
    print(self.run_thread)
    print('createThread():', self.run_thread.isAlive())

<terminated> C:\Eclipse_NEON_workspace\2nd Edition Python GUI Programming Cookbook\Ch06_Code\GUI
Hi, how are you?
<Thread(Thread-1, started 11304)>
createThread(): True
Exception in thread Thread-1:
Traceback (most recent call last):
  File "C:\Python36\lib\threading.py", line 916, in bootstrap_inner
    self.run()
  File "C:\Python36\lib\threading.py", line 864, in run
    self._target(*self._args, **self._kwargs)
  File "C:\Eclipse_NEON_workspace\2nd Edition Python GUI Programming Cookbook\Ch06"
    self.scrol.insert(tk.INSERT, str(idx) + '\n')
  File "C:\Python36\lib\tkinter\_init_.py", line 3266, in insert
    self.tk.call((self._w, 'insert', index, chars) + args)
RuntimeError: main thread is not in main loop
```

Ci si aspetta che i thread finiscano il compito assegnato, quindi quando chiudiamo la GUI mentre il thread non è stato completato, Python ci dice che il thread che abbiamo avviato non è nel ciclo di eventi principale.

Possiamo risolverlo trasformando il thread in un *demone*, che verrà quindi eseguito come attività in background.

Ciò che questo ci dà è che non appena chiudiamo la nostra GUI, che è il nostro thread principale che avvia altri thread, i thread del demone usciranno in modo pulito.

Possiamo farlo chiamando il setDaemon(Vero) metodo sul thread prima di iniziare il thread:

```
# Metodi in esecuzione nei thread
def create_thread(self):
    self.run_thread = Thread(target=self.method_in_a_thread, args=[8])
    self.run_thread.setDaemon(True)
    self.run_thread.start()
    print(self.run_thread)
```

Quando ora facciamo clic sul pulsante e usciamo dalla nostra GUI mentre il thread non ha ancora completato l'attività assegnata, non riceviamo più errori:

GUI_multiple_threads_stopping_a_thread.py

The screenshot shows the Eclipse IDE interface. On the left, there is a code editor window titled '# Running methods in Threads' containing the provided Python code. On the right, there is a 'Console' window with the following output:

```
<terminated> C:\Eclipse_NEON_workspace\2nd Edition Python GUI Programming Cookbook\Ch06_Code\GUI
Hi, how are you?
<Thread(Thread-1, started daemon 12264)>
createThread(): True
```

Come funziona...

Sebbene esista un metodo di avvio per eseguire i thread, sorprendentemente non esiste un metodo di arresto equivalente.

In questa ricetta, stiamo eseguendo un metodo in un thread, che stampa i numeri sul nostro Testo scorrevole aggeggio.

Quando usciamo dalla nostra GUI, non siamo più interessati al thread utilizzato per stampare sul nostro widget, quindi trasformando il thread in un demone, possiamo uscire dalla nostra GUI in modo pulito.

Come usare le code

Una coda Python è una struttura dati che implementa il paradigma first-in-first-out, fondamentalmente funzionando come una pipe. Metti qualcosa nel tubo da un lato e cade dall'altro lato del tubo.

La differenza principale tra questa coda che spala e che spala fango nei tubi fisici è che, nelle code Python, le cose non si confondono. Metti un'unità dentro e quell'unità torna fuori dall'altra parte. Successivamente, inserisci un'altra unità (diciamo, ad esempio, un'istanza di una classe) e l'intera unità tornerà dall'altra parte come un pezzo integrale.

Ritorna all'altra estremità nell'esatto ordine in cui abbiamo inserito il codice nella coda.



Una coda non è uno stack in cui inseriamo e inseriamo dati. Una pila è un**Last In First Out (LIFO)** struttura dati.

Le code sono contenitori che contengono dati inseriti nella coda da origini dati potenzialmente diverse. Possiamo avere diversi client che forniscono dati alla coda ogni volta che tali client hanno dati disponibili. Qualunque client sia pronto per inviare dati alla nostra coda, li invia e possiamo quindi visualizzare questi dati in un widget o inviarli ad altri moduli.

L'utilizzo di più thread per completare le attività assegnate in una coda è molto utile quando si ricevono i risultati finali dell'elaborazione e li si visualizza. I dati vengono inseriti ad un'estremità della coda e poi escono dall'altra estremità in modo ordinato, **Primo In Primo Fuori (FIFO)**.

La nostra GUI potrebbe avere cinque diversi widget di pulsanti in modo che ognuno avvia un'attività diversa, che vogliamo visualizzare nella nostra GUI in un widget (ad esempio, un Testo scorrevole aggeggio). Queste cinque diverse attività richiedono una quantità di tempo diversa per essere completate.

Ogni volta che un'attività è stata completata, dobbiamo immediatamente saperlo e visualizzare queste informazioni nella nostra GUI. Creando una coda Python condivisa e facendo scrivere i risultati alle cinque attività su questa coda, possiamo visualizzare il risultato di qualsiasi attività sia stata completata immediatamente, utilizzando l'approccio FIFO.

Prepararsi

Poiché la nostra GUI è in costante aumento in termini di funzionalità e utilità, inizia a parlare con reti, processi e siti Web e alla fine dovrà attendere che i dati siano resi disponibili per la visualizzazione da parte della GUI.

La creazione di code in Python risolve il problema dell'attesa che i dati vengano visualizzati all'interno della nostra GUI.

Come farlo...

Per creare code in Python, dobbiamo importare il Coda classe dal coda modulo. Aggiungi la seguente dichiarazione nella parte superiore del nostro modulo GUI:

```
from threading import Thread dalla  
coda import Queue
```

Questo ci fa iniziare.

Successivamente, creiamo un'istanza di coda:

```
def use_queues(self):  
    gui_queue = Coda() # crea un'istanza di coda  
    print(gui_queue) # stampa istanza
```

Chiamiamo il metodo all'interno del nostro evento clic del pulsante:

```
# Pulsante di richiamata  
def click_me(self):  
    self.action.configure(text='Hello ' + self.name.get()) self.create_thread()  
  
    self.use_queues()
```



Nel codice precedente, creiamo un local Coda istanza accessibile solo all'interno di questo metodo. Se desideriamo accedere a questa coda da altri posti, dobbiamo trasformarla in un membro della nostra classe utilizzando il pulsantese stesso parola chiave, che lega la variabile locale all'intera classe, rendendola disponibile da qualsiasi altro metodo all'interno della nostra classe. In Python, creiamo spesso variabili di istanza di classe nel __init__ se stesso) metodo, ma Python è molto pragmatico e ci consente di creare quelle variabili membro in qualsiasi punto del codice.

Ora abbiamo un'istanza di una coda. Possiamo dimostrarlo stampandolo:

The screenshot shows the Eclipse IDE interface with the 'Console' tab selected. The output window displays the following text:

```
C:\Eclipse_NEON_workspace\2nd Edition Python GUI Programming Cookbook\Ch06_Code\GUI_queues.py
Hi, how are you?
<Thread(Thread-1, started daemon 6432)>
createThread(): True
<queue.Queue object at 0x0000023C005534A8>
method_in_a_thread(): True
```

Per mettere i dati nella coda, usiamo il `mettere` comando. Per estrarre i dati dalla coda, usiamo il `ottenere` comando:

```
# Crea istanza di coda
def use_queues(self):
    gui_queue = Coda()
    print(gui_queue)
    gui_queue.put('Messaggio da una coda')
    print(gui_queue.get())
```

L'esecuzione del codice modificato fa sì che il messaggio venga prima inserito nel Coda, poi essere portato fuori dal Coda, e quindi essere stampato sulla console:

The screenshot shows the Eclipse IDE interface with the 'Console' tab selected. The output window displays the following text, corresponding to the modified code above:

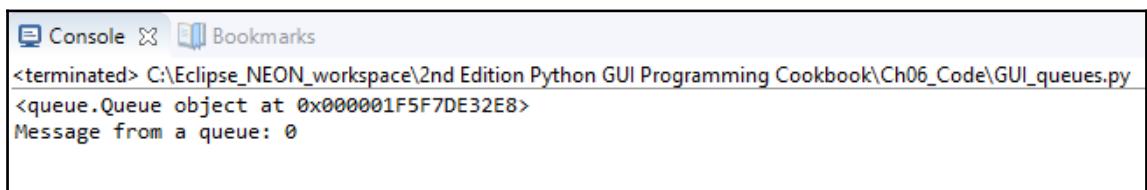
```
# Create Queue instance
def use_queues(self):
    gui_queue = Queue()
    print(gui_queue)
    gui_queue.put('Message from a queue')
    print(gui_queue.get())

<terminated> C:\Eclipse_NEON_workspace\2nd Edition Python GUI
<queue.Queue object at 0x000001B585C832B0>
Message from a queue
```

Possiamo inserire molti messaggi nel Coda:

```
# Crea istanza di coda
def use_queues(self):
    gui_queue = Coda()
    print(gui_queue)
    per idx nell'intervallo (10):
        gui_queue.put('Messaggio da una coda: ' + str(idx))
        print(gui_queue.get())
```

Abbiamo inserito dieci messaggi in Coda, ma stiamo solo tirando fuori il primo. Gli altri messaggi sono ancora dentroCoda, in attesa di essere tirato fuori in modo FIFO:



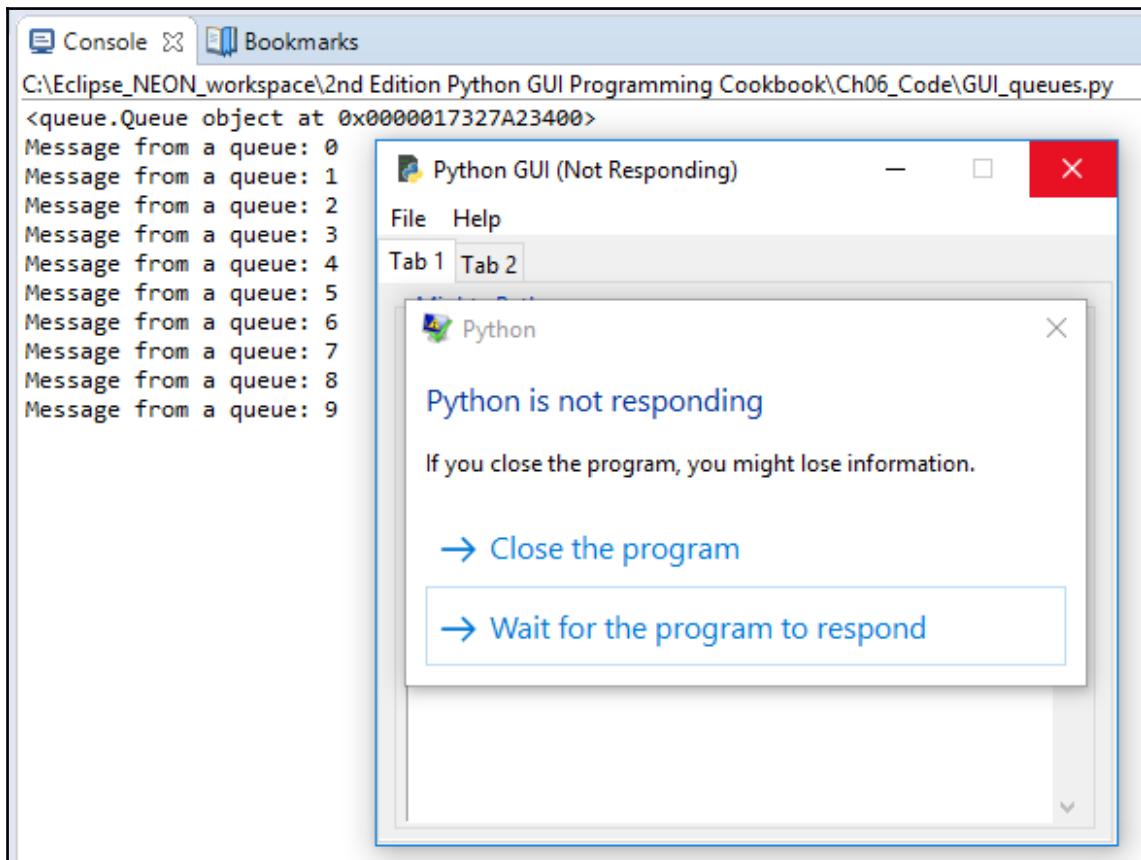
```
Console Bookmarks
<terminated> C:\Eclipse_NEON_workspace\2nd Edition Python GUI Programming Cookbook\Ch06_Code\GUI_queues.py
<queue.Queue object at 0x000001F5F7DE32E8>
Message from a queue: 0
```

Per ottenere tutti i messaggi che sono stati inseriti in Coda out, possiamo creare un ciclo infinito:

GUI_queues_put_get_loop_endless.py

```
# Crea istanza di coda
def use_queues(self):
    gui_queue = Coda()
    print(gui_queue)
    per idx nell'intervallo (10):
        gui_queue.put('Messaggio da una coda: ' + str(idx)) mentre True:
            print(gui_queue.get())
```

L'esecuzione del codice precedente risulta nella schermata seguente:



Mentre questo codice funziona, sfortunatamente, blocca la nostra GUI. Per risolvere questo problema, dobbiamo chiamare il metodo nel proprio thread, come abbiamo fatto nelle ricette precedenti.

Corriamo il nostro Coda metodo in un thread:

```
# Metodi in esecuzione nei thread
def create_thread(self):
    self.run_thread = Thread(target=self.method_in_a_thread, args=
                             [8])
    self.run_thread.setDaemon(Vero)
    self.run_thread.start()

    # avvia la coda nel proprio thread
    write_thread = Thread(target=self.use_queues, demone=True)
```

```
        write_thread.start()

# Pulsante di richiamata
def click_me(self):
    self.action.configure(text='Hello ' + self.name.get()) self.create_thread()

    # ora avviato come thread in create_thread()
    # self.use_queues()
```

Quando ora facciamo clic sul pulsante di azione, la GUI non si blocca più e il codice funziona:

GUI_queues_put_get_loop_endless_threaded.py

The screenshot shows the Eclipse IDE interface. On the left is the Python code editor with the file content. On the right is a terminal window showing the execution results.

```
# Running methods in Threads
def create_thread(self):
    self.run_thread = Thread(target=self.method_in_a_thread, args=[8])
    self.run_thread.setDaemon(True)
    self.run_thread.start()

    # start queue in its own thread
    write_thread = Thread(target=self.use_queues, daemon=True)
    write_thread.start()

# Button callback
def click_me(self):
    self.action.configure(text='Hello ' + self.name.get())
    self.create_thread()
    # self.use_queues()      # now started as a thread in create_thread()
```

Console output:

```
<terminated> C:\Eclipse_NEON_workspace\2nd Edition Python GUI Programming Cookbook\Ch06_Code\GUI_<queue.Queue object at 0x00000195FEB013C8>
Message from a queue: 0
Message from a queue: 1
Message from a queue: 2
Message from a queue: 3
Message from a queue: 4
Message from a queue: 5
Message from a queue: 6
Message from a queue: 7
Message from a queue: 8
Message from a queue: 9
```

Come funziona...

Abbiamo creato un Coda e messo i messaggi in un lato del Coda in stile FIFO. Abbiamo ottenuto i messaggi dalCoda e poi li ha stampati sulla console (stdout).

Ci siamo resi conto che dobbiamo chiamare il metodo nel proprio thread perché, altrimenti, la nostra GUI potrebbe bloccarsi.

Passaggio di code tra moduli diversi

In questa ricetta, passeremo le code attorno a diversi moduli. Man mano che il nostro codice GUI aumenta di complessità, vogliamo separare i componenti della GUI dalla logica aziendale, separandoli in moduli diversi.

La modularizzazione ci consente di riutilizzare il codice e rende anche il codice più leggibile.

Una volta che i dati da visualizzare nella nostra GUI provengono da diverse fonti di dati, affronteremo problemi di latenza, che è ciò che le code risolvono. Passando istanze diCoda tra diversi moduli Python, stiamo separando le diverse preoccupazioni delle funzionalità dei moduli.



Il codice della GUI dovrebbe idealmente riguardare solo la creazione e la visualizzazione di widget.

Il compito dei moduli di business logic è solo quello di eseguire la business logic.

Dobbiamo combinare i due elementi, idealmente utilizzando il minor numero possibile di relazioni tra i diversi moduli, riducendo l'interdipendenza del codice.



Il principio di codifica per evitare dipendenze non necessarie viene solitamente chiamato *accoppiamento lasco*.

Per comprendere il significato dell'accoppiamento lasco, possiamo disegnare alcune scatole su una lavagna o un pezzo di carta. Una casella rappresenta la nostra classe e il codice della GUI, mentre le altre caselle rappresentano la logica aziendale, i database e così via.

Successivamente, tracciamo delle linee tra i riquadri, tracciando graficamente le interdipendenze tra quei riquadri che sono i nostri moduli Python.



Meno linee abbiamo tra le nostre scatole Python, più il nostro design è debolmente accoppiato.

Prepararsi

Nella ricetta precedente, *Come utilizzare le code*, abbiamo iniziato a usare le code. In questa ricetta, passeremo istanze di aCoda dal nostro thread principale della GUI ad altri moduli Python, che ci consentiranno di scrivere nel Testo scorrevole widget da un altro modulo mantenendo la nostra GUI reattiva.

Come farlo...

Innanzitutto, creiamo un nuovo modulo Python nel nostro progetto. chiamiamoloCode.py. Ci inseriremo una funzione (non è ancora necessario l'OOP).

Passiamo un'auto-referenza della classe che crea il modulo GUI e i widget, che ci consente di utilizzare tutti i metodi della GUI da un altro modulo Python.

Lo facciamo nel callback del pulsante.



Questa è la magia dell'OOP. Nel mezzo di una classe, passiamo a una funzione che stiamo chiamando all'interno della classe, usando ilse stesso parola chiave.

Il codice ora ha il seguente aspetto:

```
import Ch06_Code.Queues come bq

classe OOP():
    # Pulsante di richiamata
    def click_me(self):
        # Passaggio nell'istanza della classe corrente (self)
        stampa (te stesso)
        bq.write_to_scrol(self)
```

Il modulo importato contiene la funzione che chiamiamo:

```
def write_to_scrol(inst):
    print('ciao dalla coda', inst)
    inst.create_thread(6)
```

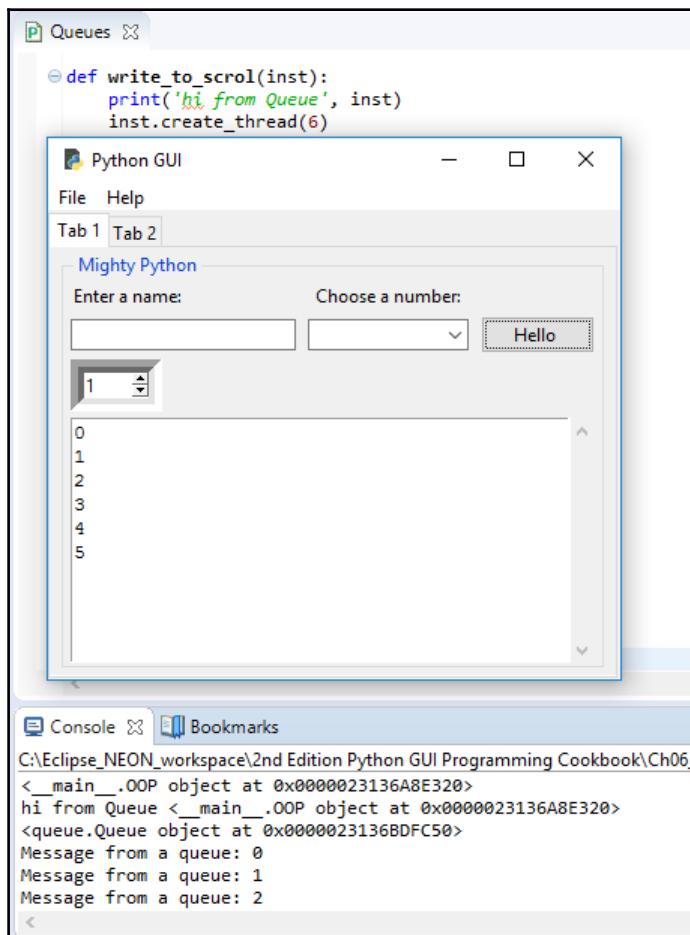
Abbiamo commentato la chiamata a `create_thread` nel callback del pulsante perché ora lo stiamo chiamando dal nostro nuovo modulo:

```
# Il metodo thread non blocca la nostra GUI  
# self.create_thread()
```



Passando un'autoreferenza dall'istanza della classe alla funzione che la classe sta chiamando in un altro modulo, ora abbiamo accesso a tutti i nostri elementi della GUI da altri moduli Python.

L'esecuzione del codice produce il seguente risultato:



Successivamente, creeremo il Coda come membro della nostra classe, ponendovi un riferimento nel `__dentro__` metodo della classe:

```
classe OOP():
    def __init__(self):
        # Crea una coda
        self.gui_queue = Coda()
```

Ora, possiamo mettere i messaggi nella coda dal nostro nuovo modulo semplicemente usando il riferimento di classe passato alla nostra GUI:

```
def write_to_scrol(inst):
    print('ciao dalla coda', inst) per idx in
    range(10):
        inst.gui_queue.put('Messaggio da una coda: ' + str(idx))
    inst.create_thread(6)
```

Il `create_thread` Il metodo nel nostro codice GUI ora legge solo dal Coda, che è stato compilato dalla logica aziendale che risiede nel nostro nuovo modulo, che ha separato la logica dal nostro modulo GUI:

```
def use_queues(self):
    # Ora usando un membro della classe Queue
    mentre vero:
        print(self.gui_queue.get())
```

L'esecuzione del nostro codice modificato produce gli stessi risultati. Non abbiamo rotto nulla (ancora)!

Come funziona...

Per separare i widget della GUI dalla funzionalità che esprime la logica di business, abbiamo creato una classe, fatto una coda un membro di questa classe e passando un'istanza della classe in una funzione che risiede in un modulo Python diverso, ora avere accesso a tutti i widget della GUI e alla coda.

Questa ricetta è un esempio di quando ha senso programmare in OOP.

Utilizzo dei widget di dialogo per copiare i file nella rete

Questa ricetta ci mostra come copiare file dal disco rigido locale in un percorso di rete.

Lo faremo utilizzando una delle finestre di dialogo integrate in tkinter di Python che ci consente di esplorare il nostro disco rigido. Possiamo quindi selezionare un file da copiare.

Questa ricetta ci mostra anche come rendere i widget di Entry di sola lettura e impostare Entry in una posizione specificata, il che accelera la navigazione del nostro disco rigido.

Prepararsi

noi estenderemo **Scheda 2** della GUI che stavamo costruendo nella ricetta precedente, *Passare le code tra i diversi moduli*.

Come farlo...

Aggiungi il seguente codice alla nostra GUI nel create_widgets() metodo verso il basso dove abbiamo creato Tab Control 2. Il genitore del frame newwidget è scheda2, che abbiamo creato all'inizio del create_widgets() metodo. Finché inserisci fisicamente il seguente codice sotto la creazione discheda2, Funzionerà:

```
#####
def create_widgets(self):
    # Crea controllo scheda
    tabControl = ttk.Notebook(self.win)
    # Aggiungi una seconda scheda
    tab2 = ttk.Frame(tabControl)
    # Rendi visibile la seconda scheda
    tabControl.add(tab2, text='Tab 2')

    # Crea una cornice di gestione dei file
    mngFilesFrame = ttk.LabelFrame(tab2, text=' Manage Files: ')
    mngFilesFrame.grid(column=0, row=1, sticky='WE', padx=10, pady=5)

    # Pulsante Richiamata
    def getFileName():
        print('ciao da getFileName')

    # Aggiungi widget per gestire la cornice dei file
    lb = ttk.Button(mngFilesFrame, text="Sfoglia nel file...",
```

```
        comando=fileName)
lb.grid(column=0, riga=0, appiccicoso=tk.W)

file = tk.StringVar()
self.entryLen = scroll_w
self.fileEntry = ttk.Entry(mngFilesFrame, width=self.entryLen,
                           variabile di testo = file)
self.fileEntry.grid(column=1, row=0, sticky=tk.W)

logDir = tk.StringVar()
self.netwEntry = ttk.Entry(mngFilesFrame,
                           larghezza=self.entryLen,
                           textvariable=logDir)
self.netwEntry.grid(column=1, riga=1, appiccicoso=tk.W)

def copyFile():
    import shutil
    src = self.fileEntry.get() file =
    src.split('/')[-1]
    dst = self.netwEntry.get() + "file prova:

    shutil.copy(src, dst)
    msg.showinfo('Copia file in rete', 'Successo:
        File copiato.')
    tranne FileNotFoundError come err:
        msg.showerror('Copia file in rete',
                      '*** Impossibile copiare il file! ***\n\n' + str(err))

    tranne Eccezione come ad es.:
        msg.showerror('Copia file in rete',
                      '*** Impossibile copiare il file! ***\n\n' + str(ex))

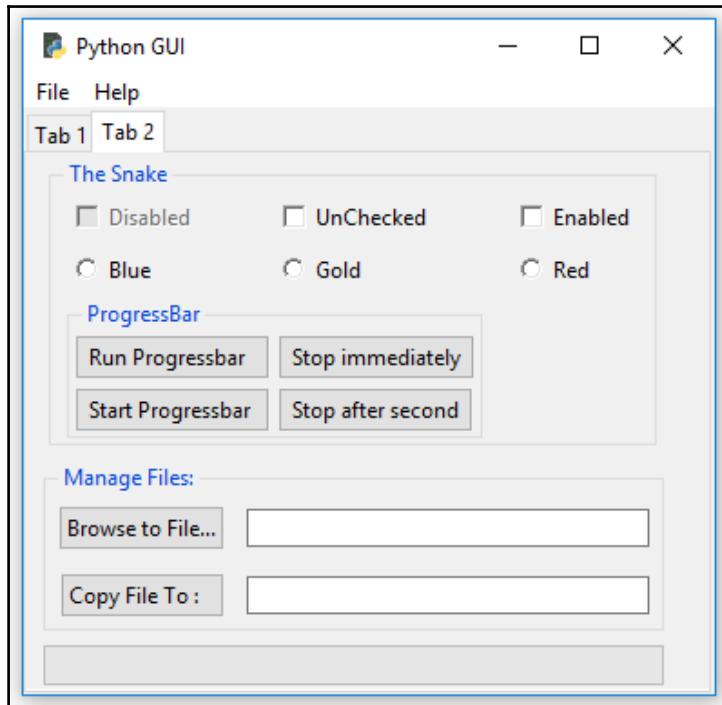
cb = ttk.Button(mngFilesFrame, text="Copia file in:",
                comando=copiaFile)
cb.grid(column=0, riga=1, sticky=tk.E)

# Aggiungi dello spazio attorno a ciascuna etichetta
per bambino in mngFilesFrame.winfo_children():
    child.grid_configure(padx=6, pady=6)
```

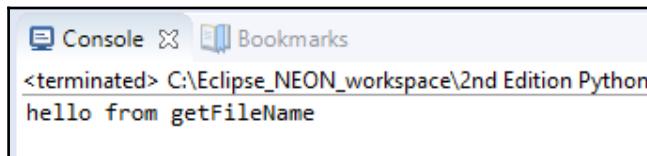
Questo aggiungerà due pulsanti e due voci a **Scheda 2** della nostra GUI.

Non stiamo ancora implementando la funzionalità della nostra funzione di callback del pulsante.

L'esecuzione del codice crea la seguente GUI:



Facendo clic su **Sfoglia file...** il pulsante attualmente stampa sulla console:



Possiamo usare le finestre di dialogo dei file integrate di tkinter, quindi aggiungiamo quanto segue importare istruzioni all'inizio del nostro modulo GUI Python:

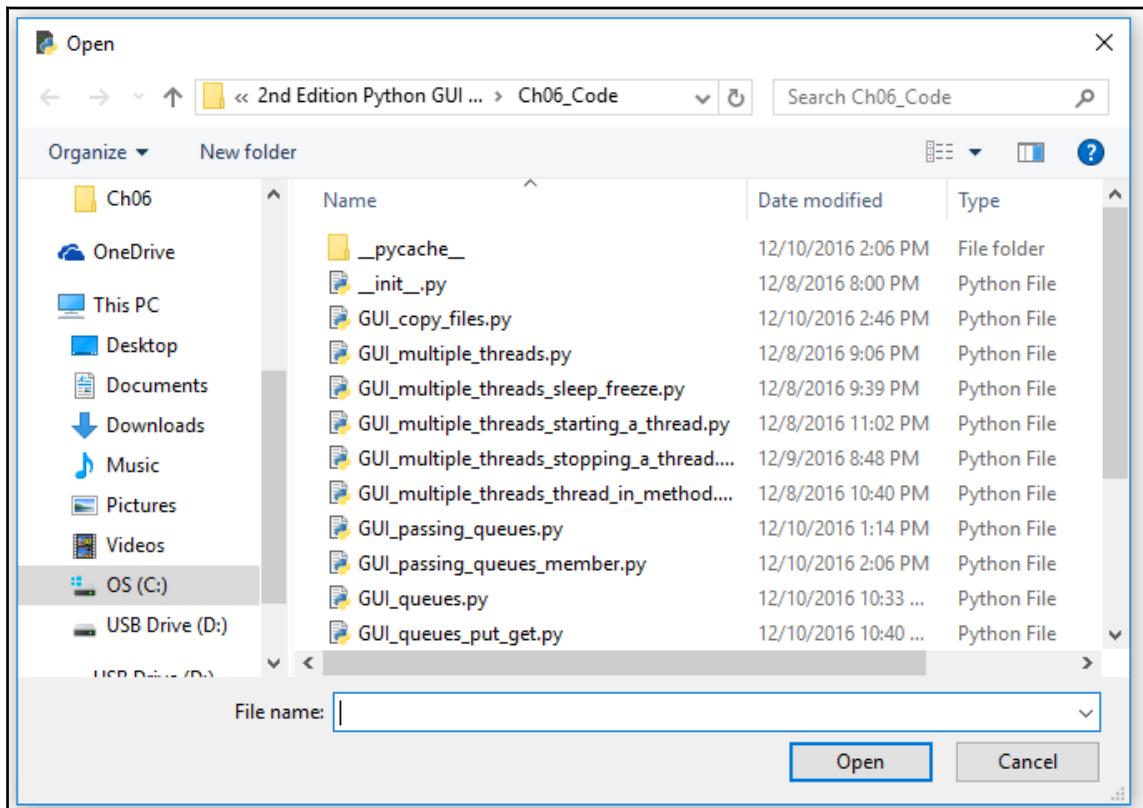
```
from tkinter import filedialog as fd from os  
import path
```

Ora possiamo usare le finestre di dialogo nel nostro codice. Invece di codificare un percorso, possiamo usare Python os module per trovare il percorso completo in cui risiede il nostro modulo GUI:

```
def getFileNome():  
    print('ciao da getFileNome')
```

```
fDir = percorso.nomedir(__file__)
fName = fd.askopenfilename(parent=self.win, initialdir=fDir)
```

Facendo clic sul pulsante Sfoglia ora si apre il askopenfilename dialogo:



Ora possiamo aprire un file in questa directory o navigare in una directory diversa. Dopo aver selezionato un file e aver fatto clic su **Aperto** pulsante nella finestra di dialogo, salveremo il percorso completo del file nel fName variabile locale.

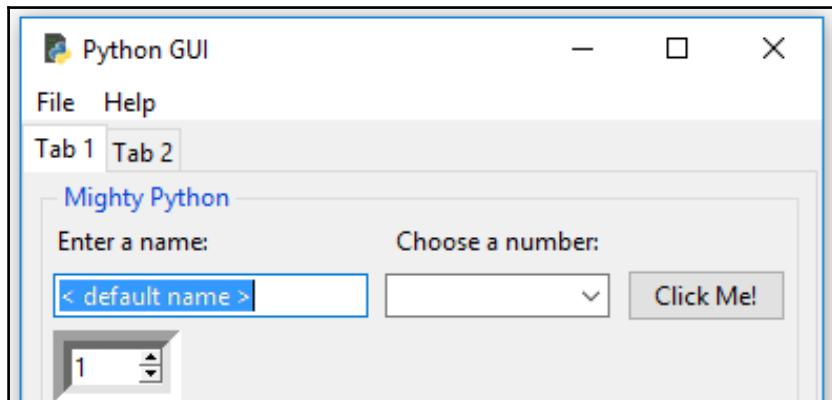
Sarebbe bello se, quando abbiamo aperto il nostro Python askopenfilename widget di dialogo, verremmo automaticamente impostati su una directory in modo da non dover navigare fino al punto in cui stavamo cercando un particolare file da aprire.

È meglio dimostrare come farlo tornando alla nostra GUI **Scheda 1**, che è quello che faremo dopo.

Possiamo impostare di default i valori nei widget Entry. Di nuovo sul nostro**Scheda 1**, questo è molto facile. Tutto quello che dobbiamo fare è aggiungere le seguenti due righe di codice alla creazione del widget Entry:

```
# Aggiunta di un widget Voce casella di testo
self.name = tk.StringVar()
self.name_entered = ttk.Entry(mighty, width=24, textvariable=self.name)
self.name_entered.grid(column=0, row=1, sticky='W')
self.name_entered.delete(0, tk.END)
self.name_entered.insert(0, '< nome predefinito >')
```

Quando ora eseguiamo la GUI, il nome_inserito la voce ha un valore predefinito:



Possiamo ottenere il percorso completo del modulo che stiamo utilizzando con la seguente sintassi Python, e quindi possiamo creare una nuova sottocartella appena sotto di essa. Possiamo farlo come globale a livello di modulo, oppure possiamo creare la sottocartella all'interno di un metodo:

```
# Livello del modulo GLOBALI
GLOBAL_CONST = 42
fDir = percorso.dirname(__file__)
netDir =
fDir + 'Backup'

def __init__(self):
    self.createWidgets()
    self.defaultFileEntries()

def defaultFileEntries(self):
    self.fileEntry.delete(0, tk.END)
    self.fileEntry.insert(0, fDir) if len(fDir) >
    self.entryLen:
        self.fileEntry.config(width=len(fDir) + 3)
        self.fileEntry.config(state='readonly')
```

```
self.netwEntry.delete(0, tk.END)
self.netwEntry.insert(0, netDir) if
len(netDir) > self.entryLen:
    self.netwEntry.config(width=len(netDir) + 3)
```

Impostiamo i valori predefiniti per entrambi i widget Entry e, dopo averli impostati, rendiamo di sola lettura il widget Entry del file locale.

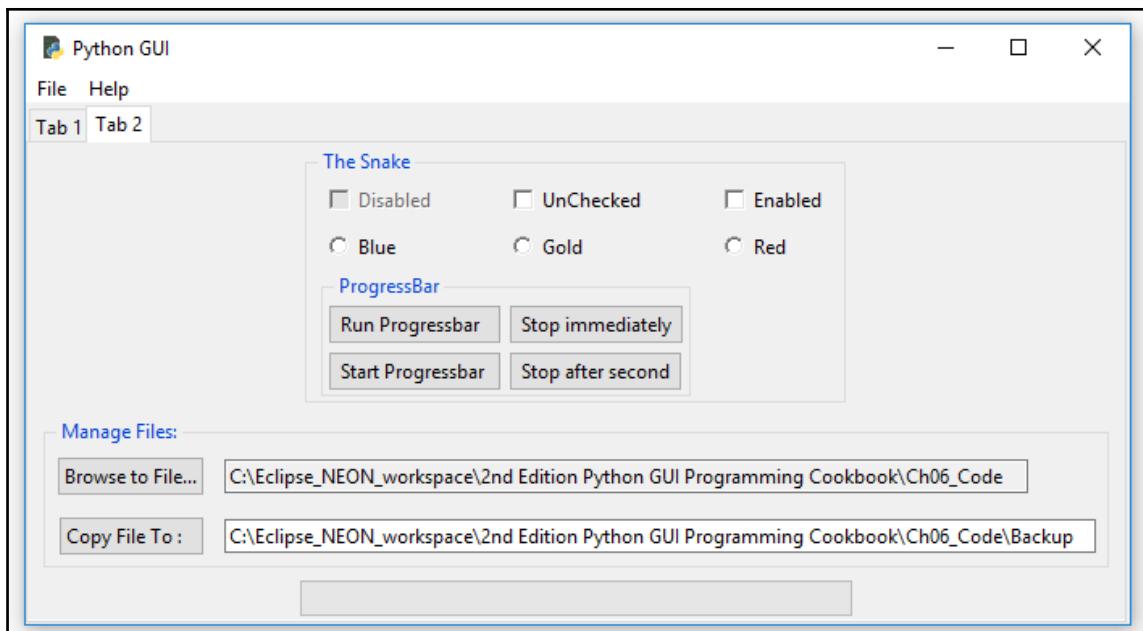


Questo ordine è importante. Dobbiamo prima popolare la voce prima di renderla di sola lettura.

Stiamo anche selezionando **Scheda 2** prima di chiamare il ciclo dell'evento principale e non impostare più lo stato attivo nella voce di **Scheda 1**. Chiamando Selezione sul nostro tkinter taccuino è in base zero, quindi passando il valore di 1, selezioniamo **Scheda 2**:

```
# Posiziona il cursore nella voce del nome
# nome_inserito.focus()
tabControl.select(1)
```

In esecuzione GUI_copy_files.py risultati nella seguente schermata:



Poiché non siamo tutti sulla stessa rete, questa ricetta utilizzerà il disco rigido locale come esempio per una rete.

Un percorso UNC è a **Convenzione di denominazione universale (UNC)** e ciò significa che usando le doppie barre rovesciate invece delle tipiche C:\, possiamo accedere a un server in rete.



Devi solo usare l'UNC e sostituire C:\ con \\<nomeserver>\<cartella>.

Questo esempio può essere utilizzato per eseguire il backup del nostro codice in una directory di backup, che possiamo creare se non esiste utilizzando os.makedirs:

```
# Livello del modulo GLOBALI
GLOBAL_CONST = 42

from os import makedirs
fDir = path.dirname(__file__)
netDir = fDir +
    'Backup' se non path.exists(netDir):

    makedirs(netDir, exist_ok = True)
```

Dopo aver selezionato un file da copiare da qualche altra parte, importiamo Python Shutil modulo. Abbiamo bisogno del percorso completo dell'origine del file da copiare e di un percorso di rete o di una directory locale, quindi aggiungiamo il nome del file al percorso in cui lo copieremo, usandoShutil.copy.



Shutil è una notazione abbreviata per l'utilità della shell.

Forniamo anche un feedback all'utente tramite una finestra di messaggio per indicare se la copia è riuscita o meno. Per fare ciò, importarecasella dei messaggi e alias è msg.

Nel prossimo codice, mescoleremo due diversi approcci su dove posizionare le nostre istruzioni di importazione. In Python abbiamo una certa flessibilità che altri linguaggi non forniscono. In genere posizioniamo tutte le istruzioni di importazione nella parte superiore di ciascuno dei nostri moduli Python in modo che sia chiaro quali moduli stiamo importando. Allo stesso tempo, un approccio di codifica moderno consiste nel posizionare la creazione di variabili vicino alla funzione o al metodo in cui vengono utilizzate per la prima volta.

Nel codice successivo, importiamo la finestra di messaggio nella parte superiore del nostro modulo Python, ma poi importiamo anche il Shutila Modulo Python in una funzione. Perché desideriamo farlo? Funziona anche questo? La risposta è sì, funziona e lo stiamo inserendoimportare un'istruzione in una funzione perché questo è l'unico posto nel nostro codice in cui abbiamo effettivamente bisogno di questo modulo.

Se non chiamiamo mai questo metodo, non importeremo mai il modulo richiesto da questo metodo. In un certo senso, puoi vedere questa tecnica come il modello di progettazione di inizializzazione pigra. Se non ne abbiamo bisogno, non lo importiamo finché non lo richiediamo davvero nel nostro codice Python. L'idea qui è che il nostro intero codice potrebbe richiedere, diciamo, 20 moduli diversi. In fase di esecuzione, quali moduli sono realmente necessari dipende dall'interazione dell'utente. Se non chiamiamo mai il copia il file() funzione, quindi non è necessario importare chiuso.

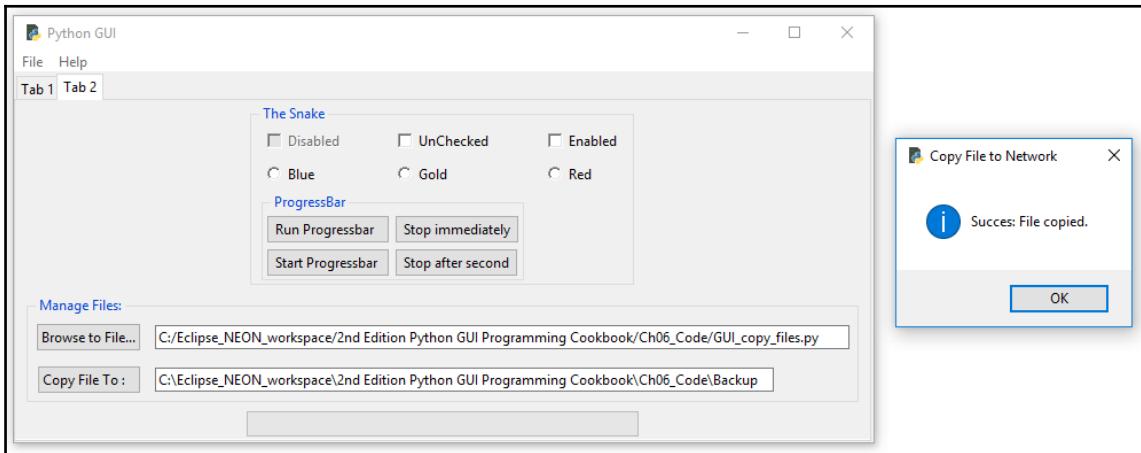
Dopo aver fatto clic sul pulsante che richiama il copia il file() funzione in questa funzione, importiamo il modulo richiesto:

```
da tkinter import messagebox come msg def
copyFile():
    import shutil                                # import modulo all'interno della funzione
    src = self.fileEntry.get() file =
    src.split('/')[-1]
    dst = self.netwEntry.get() + "file prova:

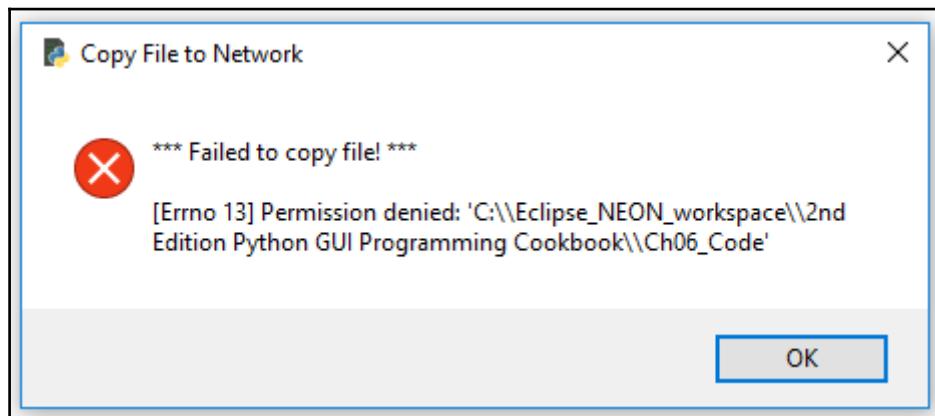
    shutil.copy(src, dst)
    msg.showinfo('Copia file in rete', 'Succes: File
copiato.')
tranne FileNotFoundError come err:
    msg.showerror('Copia file in rete',
'*** Impossibile copiare il file! ***\n\n' + str(err))

tranne Eccezione come ad es.:
    msg.showerror('Copia file in rete',
'*** Impossibile copiare il file! ***\n\n' + str(ex))
```

Quando ora eseguiamo la nostra GUI, individuare un file e fare clic su **Copia**, il file viene copiato nella posizione che abbiamo specificato nel nostro widget Entry:



Se il file non esiste o abbiamo dimenticato di cercare un file e stiamo provando a copiare l'intera cartella padre, il codice ce lo farà sapere anche perché stiamo usando le capacità di gestione delle eccezioni integrate di Python:

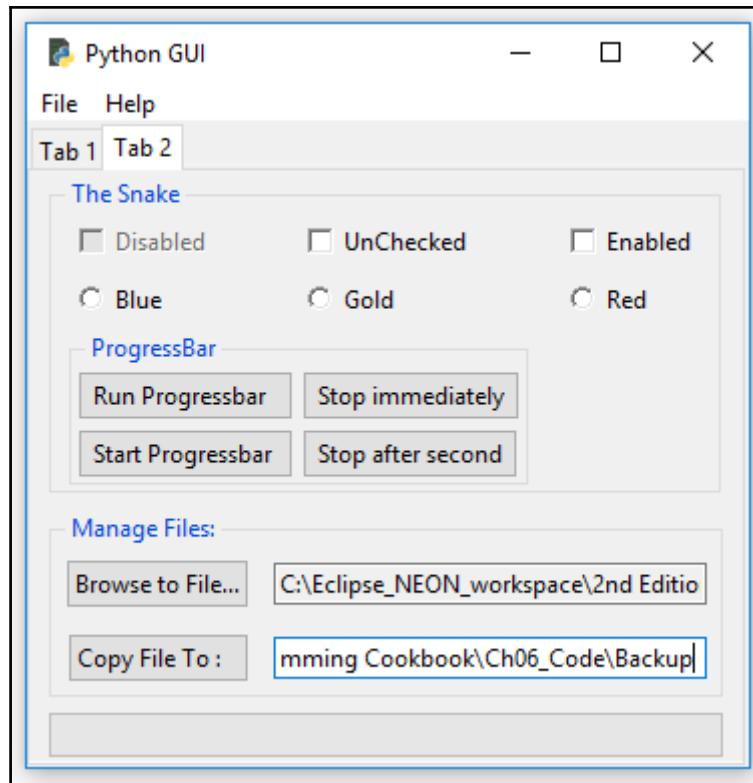


I nostri nuovi widget Entry hanno ampliato la larghezza della GUI. Mentre a volte è bello poter vedere l'intero percorso, allo stesso tempo spinge altri widget, rendendo la nostra GUI non così buona. Possiamo risolverlo limitando il parametro di larghezza dei nostri widget di immissione:

```
46     self.defaultFileEntries()
47
48     def defaultFileEntries(self):
49         self.fileEntry.delete(0, tk.END)
50         self.fileEntry.insert(0, fDir)
51         if len(fDir) > self.entryLen:
52             self.fileEntry.config(width=len(fDir) + 3)
53         # self.fileEntry.config(width=35)           # limit width to adjust GUI
54         self.fileEntry.config(state='readonly')
55
56
57         self.netwEntry.delete(0, tk.END)
58         self.netwEntry.insert(0, netDir)
59         if len(netDir) > self.entryLen:
60             self.netwEntry.config(width=len(netDir) + 3)
61         # self.netwEntry.config(width=35)           # limit width to adjust GUI
62
```

Ciò si traduce nella seguente dimensione della GUI. Possiamo usare la freccia destra nel widget Entry abilitato per arrivare alla fine di questo widget:

GUI_copy_files_limit.py



Come funziona...

Stiamo copiando i file dal nostro disco rigido locale a una rete utilizzando l'utilità della shell Python. Poiché la maggior parte di noi non è connessa alla stessa rete locale, simuliamo la copia eseguendo il backup del codice in una cartella locale diversa.

Stiamo usando uno dei controlli di dialogo di tkinter e, impostando i percorsi di directory predefiniti, possiamo aumentare la nostra efficienza nella copia dei file.

Utilizzo di TCP/IP per comunicare in rete

Questa ricetta ti mostra come usare prese per comunicare tramite TCP/IP. Per raggiungere questo obiettivo, abbiamo bisogno sia di un indirizzo IP che di un numero di porta.

Per mantenere le cose semplici e indipendenti dagli indirizzi IP Internet in continua evoluzione, creeremo il nostro server TCP/IP locale e, come client, impareremo come connetterci e leggere i dati da una connessione TCP/IP.

Integreremo questa funzionalità di rete nella nostra GUI utilizzando le code che abbiamo creato nelle ricette precedenti.



TCP/IP è l'acronimo di Transmission Control Protocol/Internet Protocol, che è un insieme di protocolli di rete che consente a due o più computer di comunicare.

Prepararsi

Creeremo un nuovo modulo Python che sarà il server TCP.

Come farlo...

Un modo per implementare un server TCP in Python è ereditare dal socketserver modulo. noi sottoclassiamo `BaseRequestHandler` e poi sovrascrivere l'ereditato `maniglia` metodo. In pochissime righe di codice Python, possiamo implementare un server TCP:

```
da socketserver import BaseRequestHandler, TCPServer

classe RequestHandler(BaseRequestHandler):
    # sovrascrive il metodo di gestione della classe base
    def handle(self):
        print('Server connesso a: ', self.client_address) mentre True:

            rsp = self.request.recv(512) if not rsp:
            break
            self.request.send(b'Server ricevuto: ' + rsp)

    def start_server():
        server = TCPServer((" ", 24000), RequestHandler)
        server.serve_forever()
```

Stiamo passando nel nostro RequestHandler classe in a TCPServer inizializzatore. Le virgolette singole vuote sono una scorciatoia per passare in localhost, che è il nostro PC. Questo è l'indirizzo IP di 127.0.0.1. Il secondo elemento nella tupla è il numero di porta. Possiamo scegliere qualsiasi numero di porta che non è in uso sul nostro PC locale.

Dobbiamo solo assicurarci di utilizzare la stessa porta sul lato client della connessione TCP; altrimenti, non saremmo in grado di connetterci al server. Ovviamente, dobbiamo avviare il server prima che i client possano connettersi ad esso.

Modificheremo il nostro Code.py modulo per diventare il client TCP:

```
# utilizzando TCP/IP
da socket import socket, AF_INET, SOCK_STREAM

def write_to_scrol(inst):
    print('ciao dalla coda', inst)
    sock = socket(AF_INET, SOCK_STREAM)
    sock.connect(('localhost', 24000)) per idx in
    range(10):
        sock.send(b'Messaggio da una coda: ' +
                  byte(str(idx).encode()))
    recv = sock.recv(8192).decode()
    inst.gui_queue.put(recv)
    inst.create_thread(6)
```

Quando ora clicchiamo su **Cliccamici!** pulsante, stiamo chiamando bq.write_to_scrol(self), che quindi crea il socket e la connessione mostrati in precedenza.

Questo è tutto il codice di cui abbiamo bisogno per parlare con il server TCP. In questo esempio, stiamo semplicemente inviando alcuni byte al server e il server li restituisce, anteponendo alcune stringhe prima di restituire la risposta.

Questo mostra il principio di come funzionano le comunicazioni TCP tramite le reti.



Una volta che sappiamo come connetterci a un server remoto tramite TCP/IP, utilizzeremo qualunque comando sia progettato dal protocollo del programma con cui siamo interessati a comunicare. Il primo passo è connettersi prima di poter inviare comandi ad applicazioni specifiche che risiedono su un server.

Nel writeToScrol funzione, utilizzeremo lo stesso ciclo di prima, ma ora invieremo i messaggi al server TCP. Il server modifica il messaggio ricevuto e poi ce lo rimanda. Successivamente lo inseriamo nella coda dei membri della GUI, che, come nelle ricette precedenti, viene eseguita nel proprio thread:

```
sock.send(b'Messaggio da una coda: ' + bytes(str(idx).encode()) )
```

Notare la b prima lo spago e poi, beh, tutto il resto del casting richiesto.

Avviamo il server TCP nel proprio thread nell'inizializzatore della classe OOP:

```
classe OOP():
    def __init__(self):
        # Avvia il server TCP/IP nel proprio thread
        svrT = Thread(target=startServer, demone=True) svrT.start()
```



In Python 3 dobbiamo inviare stringhe su socket in formato binario. L'aggiunta dell'indice intero ora diventa un po' contorta poiché dobbiamo convertirlo in una stringa, codificarlo e quindi eseguire il cast della stringa codificata in byte!

Facendo clic su **Cliccam!** pulsante acceso **Scheda 1** ora crea il seguente output nel nostro Testo scorrevole widget oltre che sulla console, e la risposta dovuta all'utilizzo dei thread è velocissima:

GUI_TCP_IP.py

The screenshot shows a Python application interface. On the left, a 'Console' window displays the following text output:

```
C:\EclipseWorkspace\Ch06new\B04829_Ch06_Code\B04829_Ch06_GUI.py
<__main__.OOP object at 0x000000002B51898>
hi from Queue <__main__.OOP object at 0x000000002B51898>
Server connected to: ('127.0.0.1', 58295)
<Thread(Thread-2, started daemon 6164)>
createThread(): True
Server received: Message from a queue: 0
Server received: Message from a queue: 1
Server received: Message from a queue: 2
Server received: Message from a queue: 3
Server received: Message from a queue: 4
Server received: Message from a queue: 5
Server received: Message from a queue: 6
Server received: Message from a queue: 7
Server received: Message from a queue: 8
Server received: Message from a queue: 9
methodInAThread(): True
```

On the right, a 'Python GUI' window titled 'Monty Python' is shown. It has two tabs: 'File' and 'Help'. The 'File' tab is selected. Inside the window, there are two input fields: 'Enter a name:' and 'Choose a number:', both currently set to 'Hello < default name >'. Below these fields is a dropdown menu with the value '1'. A scrollable list box displays the same sequence of messages as the console:

```
Server received: Message from a queue: 0
Server received: Message from a queue: 1
Server received: Message from a queue: 2
Server received: Message from a queue: 3
Server received: Message from a queue: 4
Server received: Message from a queue: 5
Server received: Message from a queue: 6
Server received: Message from a queue: 7
Server received: Message from a queue: 8
Server received: Message from a queue: 9
```

Come funziona...

Abbiamo creato un server TCP per simulare la connessione a un server nella nostra rete locale o su Internet. Abbiamo trasformato il nostro modulo delle code in un client TCP. Stiamo eseguendo sia la coda che il server nel loro thread in background, il che mantiene la nostra GUI molto reattiva.

Utilizzo di urlopen per leggere i dati dai siti Web

Questa ricetta mostra come possiamo leggere facilmente intere pagine web usando i moduli integrati di Python. Visualizzeremo i dati della pagina Web prima nel suo formato non elaborato e poi lo decodificheremo, quindi lo visualizzeremo nella nostra GUI.

Prepararsi

Leggeremo i dati da una pagina web e poi li visualizzeremo nel Testo scorrevole widget della nostra GUI.

Come farlo...

Per prima cosa, creiamo un nuovo modulo Python e lo chiamiamo URL.py. Quindi importiamo la funzionalità richiesta per leggere le pagine Web utilizzando Python. Possiamo farlo in pochissime righe di codice.

Avvolgiamo il nostro codice in a prova...tranne blocco simile a Java e C#. Questo è un approccio moderno alla codifica, supportato da Python. Ogni volta che abbiamo del codice che potrebbe non essere completo, possiamo sperimentare con questo codice e, se funziona, va tutto bene. Se il blocco di codice nel prova...tranne block non funziona, l'interprete Python genererà una delle diverse possibili eccezioni, che possiamo quindi rilevare. Una volta che abbiamo catturato l'eccezione, possiamo decidere cosa fare dopo.

Esiste una gerarchia di eccezioni in Python e possiamo anche creare le nostre classi che ereditano ed estendono le classi di eccezioni di Python. Nel codice seguente, siamo principalmente preoccupati che l'URL che stiamo cercando di aprire potrebbe non essere disponibile, quindi avvolgiamo il nostro codice all'interno di un prova...tranne blocco di codice. Se il codice riesce ad aprire l'URL richiesto, va tutto bene. Se fallisce, forse perché la nostra connessione Internet non funziona, cadiamo nella parte di eccezione del codice e stampiamo che si è verificata un'eccezione:

```
from urllib.request import urlopen link =  
'http://python.org/'
```

```
provare:  
    http_rsp = urlopen(link)  
    print(http_rsp)  
    html = http_rsp.read()  
    stampa(html)  
    html_decoded = html.decode()  
    print(html_decoded)  
tranne Eccezione come ad es.:  
    print('*** Impossibile ottenere Html! ***\n\n' + str(ex)) else:  
  
    return html_decoded
```

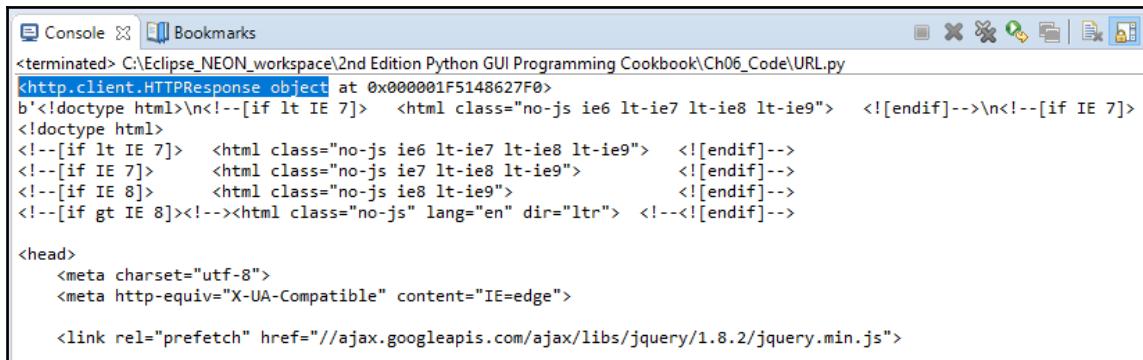


Puoi leggere di più sulla gestione delle eccezioni Python su <https://docs.python.org/3.6/library/exceptions.html>.

A chiamata urlopen sul sito Web ufficiale di Python, otteniamo tutti i dati come una lunga stringa. La prima istruzione print stampa questa lunga stringa sulla console.

poi chiamiamo decodificare sul risultato, e questa volta otteniamo poco più di 1.000 righe di dati web, inclusi alcuni spazi bianchi.

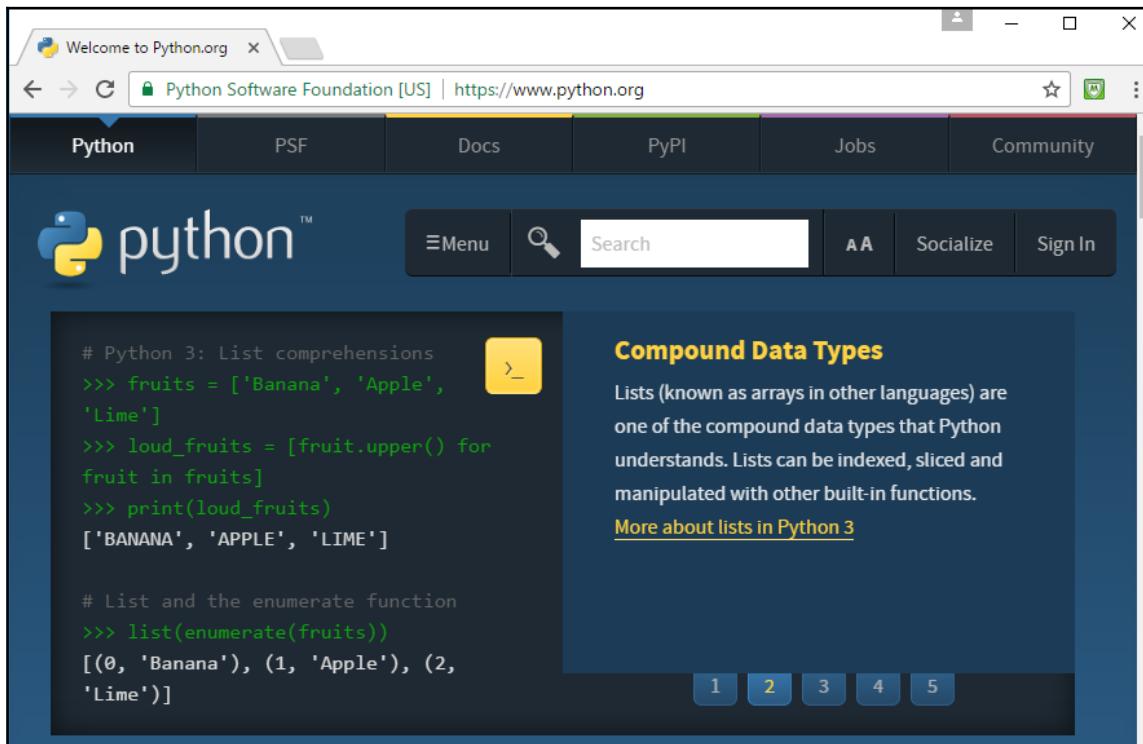
Stampiamo anche il tipo per la chiamata urlopen, che è un http.client.HTTPResponse oggetto. In realtà, lo stampiamo prima:



The screenshot shows the Eclipse IDE's Console view. The title bar says "Console Bookmarks". The content area displays the output of the Python script. It starts with the file path "C:\Eclipse_NEON_workspace\2nd Edition Python GUI Programming Cookbook\Ch06_Code\URL.py", followed by the type of the variable "khttp.client.HTTPResponse object at 0x000001F5148627F0>". The output then shows a large block of HTML code with various conditional comments for Internet Explorer versions 7, 8, and above. At the bottom, there is a standard HTML head section with meta tags for charset and http-equiv, and a link tag for a jQuery library.

```
<terminated> C:\Eclipse_NEON_workspace\2nd Edition Python GUI Programming Cookbook\Ch06_Code\URL.py  
khttp.client.HTTPResponse object at 0x000001F5148627F0>  
b'<!doctype html>\n<!--[if lt IE 7]>    <html class="no-js ie6 lt-ie7 lt-ie8 lt-ie9">    <![endif]-->\n<!--[if IE 7]>  
<!doctype html>  
<!--[if lt IE 7]>    <html class="no-js ie6 lt-ie7 lt-ie8 lt-ie9">    <![endif]-->  
<!--[if IE 7]>    <html class="no-js ie7 lt-ie8 lt-ie9">        <![endif]-->  
<!--[if IE 8]>    <html class="no-js ie8 lt-ie9">        <![endif]-->  
<!--[if gt IE 8]><!--><html class="no-js" lang="en" dir="ltr">    <!--<![endif]-->  
  
<head>  
    <meta charset="utf-8">  
    <meta http-equiv="X-UA-Compatible" content="IE=edge">  
  
    <link rel="prefetch" href="//ajax.googleapis.com/ajax/libs/jquery/1.8.2/jquery.min.js">
```

Ecco la pagina web ufficiale di Python che abbiamo appena letto. Se sei uno sviluppatore web, probabilmente hai delle buone idee su cosa fare con i dati analizzati:



Quindi mostriamo questi dati nella nostra GUI all'interno del Testo scorrevole aggeggio. Per fare ciò, dobbiamo connettere il nostro nuovo modulo, che legge i dati dalla pagina web alla nostra GUI.

Per fare ciò, abbiamo bisogno di un riferimento alla nostra GUI e un modo per farlo è legare il nostro nuovo modulo al **Scheda 1** pulsante di richiamata. Possiamo restituire i dati HTML decodificati dalla pagina Web Python al Pulsante widget, che possiamo poi inserire nel Testo scorrevole controllo.

Quindi, trasformiamo il nostro codice in una funzione e restituiamo i dati al codice chiamante:

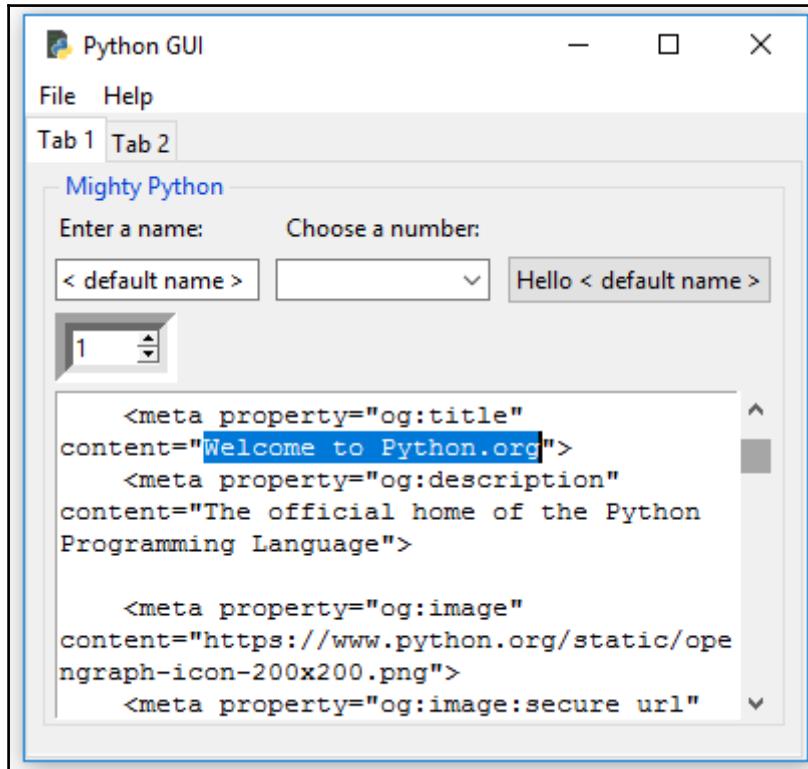
```
from urllib.request import urlopen link =  
'http://python.org/'  
  
def get_html():  
    provare:  
        http_rsp = urlopen(link)  
        print(http_rsp)  
        html = http_rsp.read()  
        stampa(html)  
        html_decoded = html.decode()  
        print(html_decoded)  
    tranne Eccezione come ad es.:  
        print('*** Impossibile ottenere Html! ***\n\n' + str(ex)) else:  
  
        return html_decoded
```

Ora possiamo scrivere i dati dal nostro metodo di callback del pulsante a Testo scorrevole controllo importando prima il nuovo modulo e poi inserendo i dati nel widget. Gli diamo anche un po' di sonno dopo la chiamata `awrite_to_scrol`:

```
importa Ch06_Code.URL come url  
  
# Pulsante di richiamata  
def click_me(self):  
    self.action.configure(text='Hello ' + self.name.get()) bq.write_to_scrol(self)  
  
    dormire(2)  
    html_data = url.get_html()  
    print(html_data)  
    self.scrol.insert(tk.INSERT, html_data)
```

I dati HTML sono ora visualizzati nel nostro widget GUI:

GUI_URL.py



Come funziona...

Creiamo un nuovo modulo per separare il codice che ottiene i dati da una pagina web dal nostro codice GUI. Questa è sempre una buona cosa da fare. Leggiamo i dati della pagina web e poi li restituiamo al codice chiamante dopo averlo decodificato. Utilizziamo quindi la funzione di callback del pulsante per inserire i dati restituiti nelTesto scorrevole controllo.

Questo capitolo ci ha introdotto ad alcuni concetti avanzati di programmazione Python, che abbiamo combinato per produrre un programma GUI funzionale.

7

Memorizzazione dei dati nel nostro MySQL Database tramite la nostra GUI

In questo capitolo miglioreremo la nostra GUI Python connettendoci a un database MySQL. Tratteremo le seguenti ricette:

- Installazione e connessione a un server MySQL da Python
- Configurazione della connessione al database MySQL
- Progettazione del database della GUI
- Python Utilizzo del comando SQL INSERT
- Utilizzo del comando SQL UPDATE
- Utilizzo del comando SQL DELETE
- Memorizzazione e recupero dei dati dal nostro database MySQL
- Utilizzando il workbench MySQL

introduzione

Prima di poterci connettere a un server MySQL, dobbiamo avere accesso a un server MySQL.



La prima ricetta in questo capitolo ti mostrerà come installare la MySQL Server Community Edition gratuita.

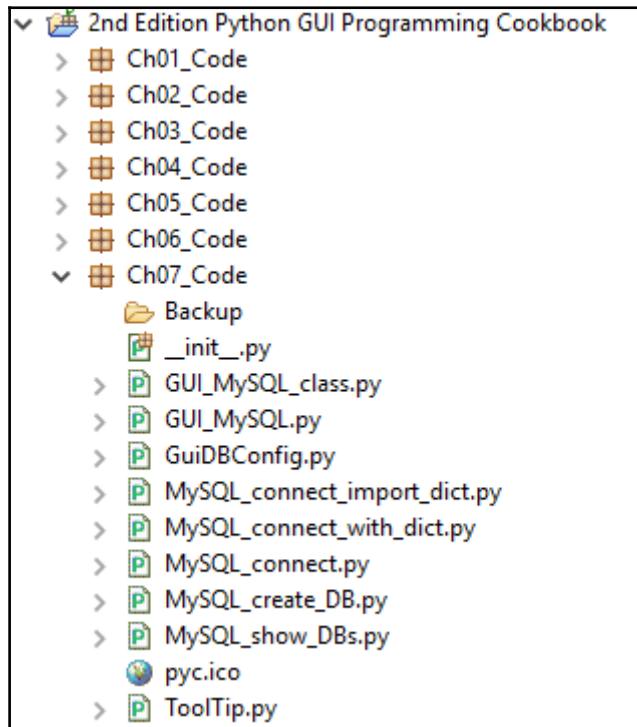
Dopo aver eseguito correttamente la connessione a un'istanza in esecuzione del nostro server MySQL, progetteremo e creeremo un database che accetterà il titolo di un libro, che potrebbe essere il nostro diario o una citazione che abbiamo trovato da qualche parte su Internet. Avremo bisogno di un numero di pagina per il libro, che potrebbe essere vuoto, e poi, lo faremo inserire la citazione che ci piace da un libro, un diario, un sito Web o un amico nel nostro database MySQL utilizzando la nostra GUI, creata utilizzando Python 3.6 e versioni successive.

Inseriremo, modificheremo, cancelleremo e mostreremo le nostre citazioni preferite usando la nostra GUI Python per emettere questi comandi SQL e per visualizzare i dati.



CRUD è un termine di database che potresti incontrare, che abbrevia i quattro comandi SQL di base e sta per **Crea, Leggi, Aggiorna, e Elimina**.

Ecco la panoramica dei moduli Python per questo capitolo:



Installazione e connessione a un server MySQL da Python

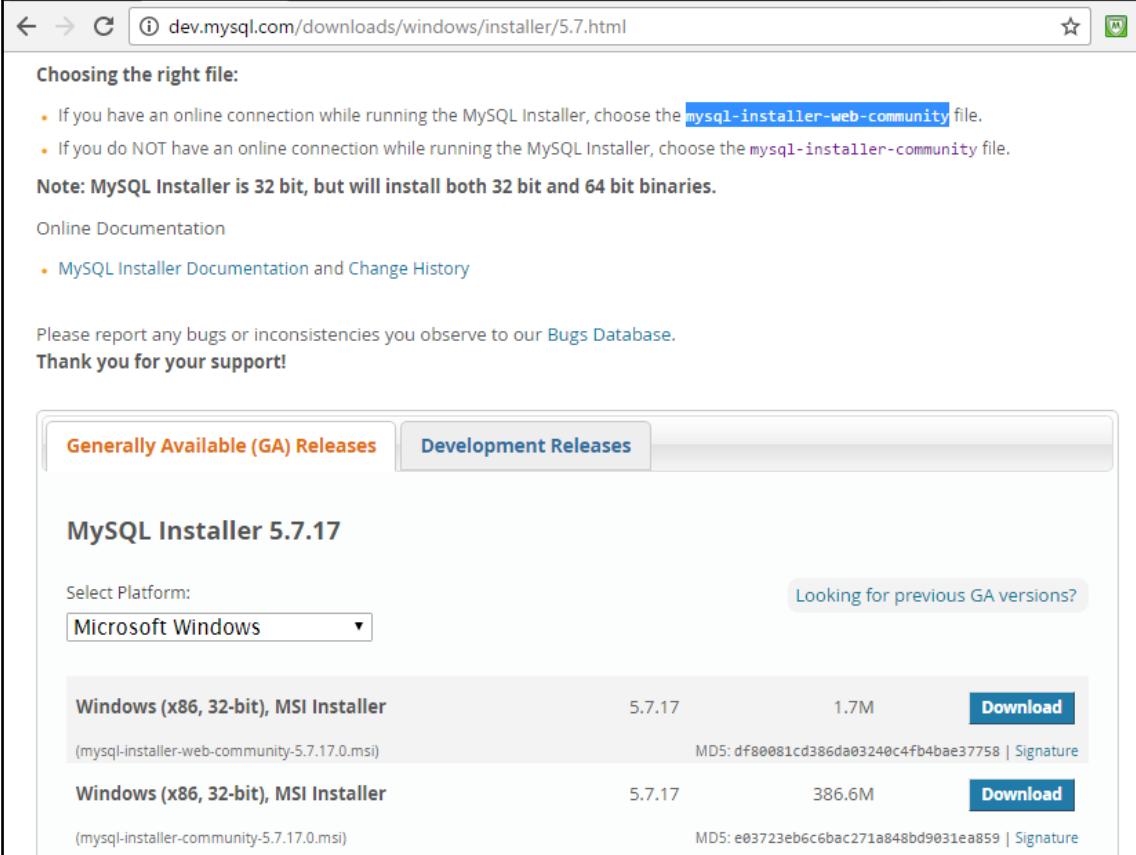
Prima di poterci connettere a un database MySQL, dobbiamo connetterci a MySQL Server. Per fare ciò, abbiamo bisogno di conoscere l'indirizzo IP del server MySQL e la porta su cui è in ascolto.

Dobbiamo anche essere un utente registrato con una password per essere autenticati dal server MySQL.

Prepararsi

Dovrai avere accesso a un'istanza di MySQL Server in esecuzione e devi anche disporre dei privilegi di amministratore per creare database e tabelle.

È disponibile una MySQL Community Edition gratuita sul sito Web ufficiale di MySQL. Puoi scaricarlo e installarlo sul tuo PC locale da <http://dev.mysql.com/downloads/windows/installer/5.7.html>:



The screenshot shows a web browser displaying the MySQL Installer download page at dev.mysql.com/downloads/windows/installer/5.7.html. The page header includes navigation icons and a search bar. Below the header, there's a section titled "Choosing the right file:" with two bullet points:

- If you have an online connection while running the MySQL Installer, choose the `mysql-installer-web-community` file.
- If you do NOT have an online connection while running the MySQL Installer, choose the `mysql-installer-community` file.

A note below states: "Note: MySQL Installer is 32 bit, but will install both 32 bit and 64 bit binaries." There are links for "Online Documentation" and "MySQL Installer Documentation and Change History". A message encourages reporting bugs to the "Bugs Database" and expresses gratitude for support.

The main content area features two tabs: "Generally Available (GA) Releases" (highlighted in orange) and "Development Releases". Below this, the title "MySQL Installer 5.7.17" is displayed. A "Select Platform:" dropdown is set to "Microsoft Windows". To the right, a link "Looking for previous GA versions?" is visible. Two download options are listed:

File Type	Version	Size	Action
Windows (x86, 32-bit), MSI Installer (mysql-installer-web-community-5.7.17.0.msi)	5.7.17	1.7M	Download
Windows (x86, 32-bit), MSI Installer (mysql-installer-community-5.7.17.0.msi)	5.7.17	386.6M	Download

MD5 checksums and signature links are provided for each download.

Durante il processo di installazione, sceglierai una password per l'utente root e potrai anche aggiungere più utenti. Ti consiglio di aggiungerti come **Amministratore DB** e scegli anche una password:

Accounts and Roles

Root Account Password

Enter the password for the root account. Please remember to store this password in a secure place.

MySQL Root Password:

Repeat Password:

Password Strength: Weak

MySQL User Accounts

Create MySQL user accounts for your users and applications. Assign a role to the user that consists of a set of privileges.

	MySQL Username	Host	User Role	
	Burkhard	%	DB Admin	Add User Edit User Delete



In questo capitolo, stiamo utilizzando l'ultima versione di MySQL Community Server 5.7.17.

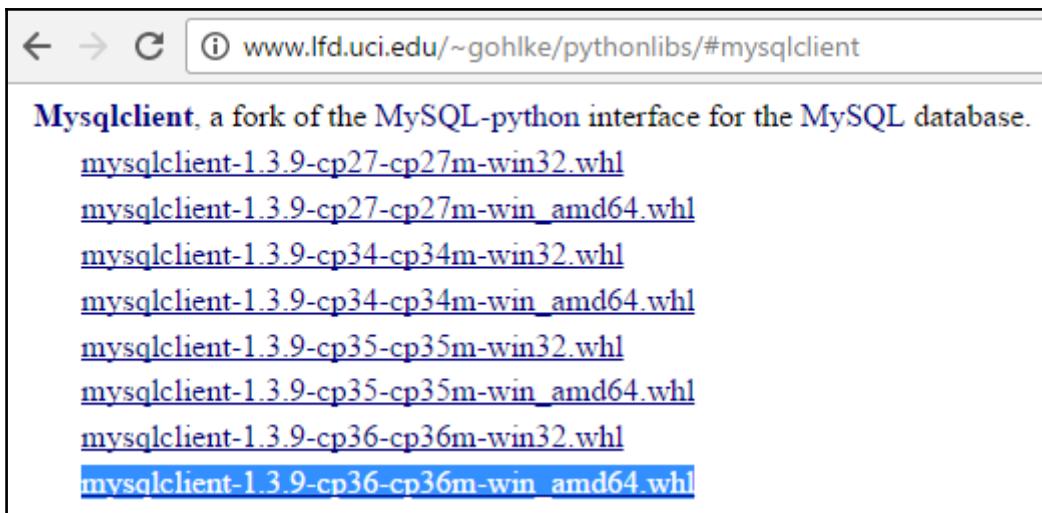
Come farlo...

Per connetterci a MySQL, dobbiamo prima installare uno speciale driver del connettore Python. Questo driver ci consentirà di parlare con il server MySQL da Python. C'è un driver disponibile gratuitamente sul sito Web di MySQL e viene fornito con un tutorial online molto carino:

<http://dev.mysql.com/doc/connector-python/en/index.html>

Al momento della stesura di questo libro, questo connettore MySQL non è stato ancora aggiornato a Python 3.6, quindi seguiremo un approccio leggermente diverso.

A partire dal <http://www.lfd.uci.edu/~gohlke/pythonlibs/#mysqlclient>, possiamo scaricare un pacchetto che ci permette di parlare con il nostro server MySQL tramite Python 3.6:

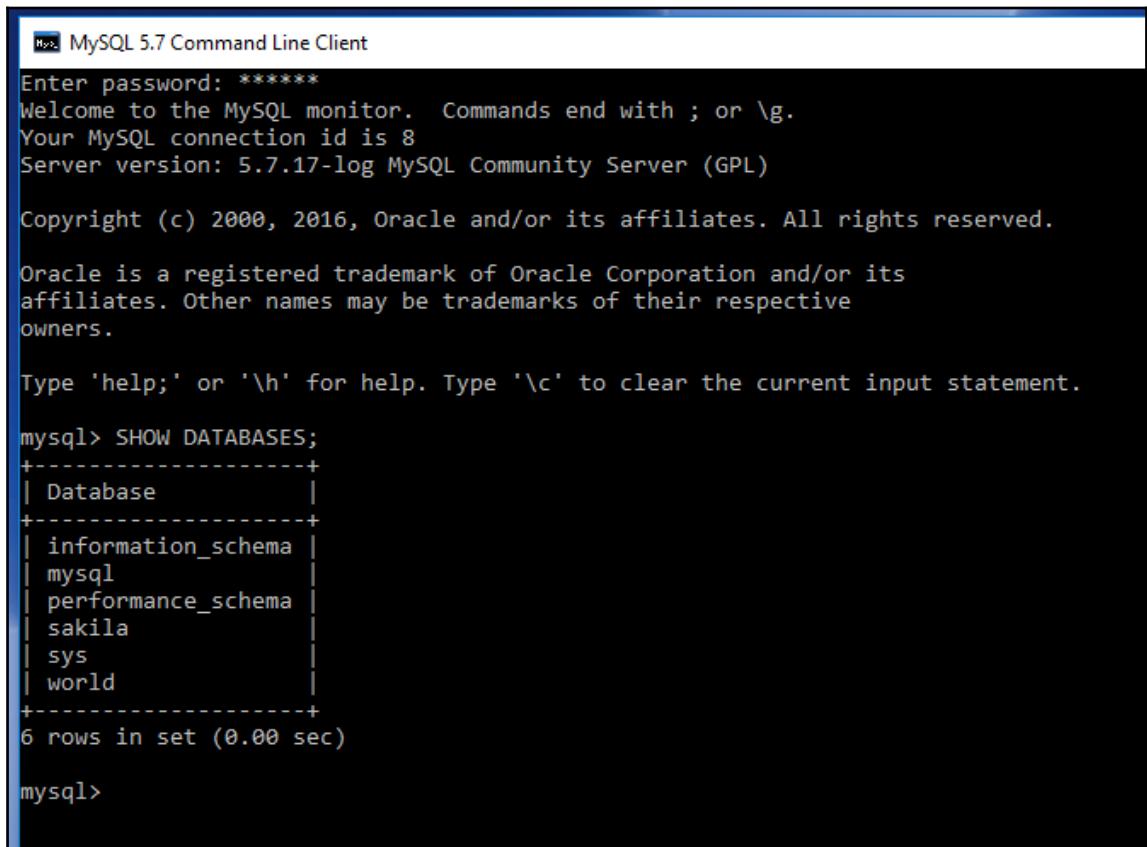


La ruota evidenziata (.whl) pacchetto di installazione corrisponde alla nostra installazione a 64 bit di Python 3.6 su un sistema operativo Windows 10 a 64 bit.

Un modo per verificare di aver installato il driver corretto e che consente a Python di parlare con MySQL, è esaminare Python pacchetti-sito directory. Se tuopacchetti-sito la directory ha una nuova MySQLdb cartella e qualche altra _mysql moduli, l'installazione è andata a buon fine:

Name	Date modified	Type	Size
<input checked="" type="checkbox"/> MySQLdb	12/13/2016 8:13 PM	File folder	
<input checked="" type="checkbox"/> numpy	12/4/2016 6:38 PM	File folder	
<input checked="" type="checkbox"/> numpy-1.11.2-py3.6.egg-info	12/4/2016 6:38 PM	File folder	
<input checked="" type="checkbox"/> pip	11/22/2016 8:45 PM	File folder	
<input checked="" type="checkbox"/> pip-9.0.1.dist-info	11/22/2016 8:45 PM	File folder	
<input checked="" type="checkbox"/> pkg_resources	11/22/2016 8:45 PM	File folder	
<input checked="" type="checkbox"/> pyparsing-2.1.10.dist-info	12/4/2016 6:36 PM	File folder	
<input checked="" type="checkbox"/> python_dateutil-2.6.0.dist-info	12/4/2016 4:48 PM	File folder	
<input checked="" type="checkbox"/> pytz	12/4/2016 6:36 PM	File folder	
<input checked="" type="checkbox"/> pytz-2016.7.dist-info	12/4/2016 6:36 PM	File folder	
<input checked="" type="checkbox"/> setuptools	11/22/2016 8:45 PM	File folder	
<input checked="" type="checkbox"/> setuptools-28.8.0.dist-info	11/22/2016 8:45 PM	File folder	
<input checked="" type="checkbox"/> six-1.10.0.dist-info	12/4/2016 4:48 PM	File folder	
<input checked="" type="checkbox"/> _mysql.cp36-win_amd64.pyd	12/13/2016 8:13 PM	Python Extension ...	3,784 KB
<input checked="" type="checkbox"/> _mysql_exceptions.py	12/13/2016 8:13 PM	Python File	3 KB

Innanzitutto, verifichiamo che l'installazione del nostro server MySQL funzioni utilizzando il *Client da riga di comando MySQL*. Al mysql> prompt, digitare MOSTRA BANCHE DATI; quindi premere *Accedere*:



```
MySQL 5.7 Command Line Client
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 8
Server version: 5.7.17-log MySQL Community Server (GPL)

Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> SHOW DATABASES;
+-----+
| Database      |
+-----+
| information_schema |
| mysql          |
| performance_schema |
| sakila         |
| sys            |
| world          |
+-----+
6 rows in set (0.00 sec)

mysql>
```

Successivamente, verificheremo che possiamo ottenere gli stessi risultati usando Python 3.6:



Sostituisci i nomi segnaposto tra parentesi <adminUser> e <adminPwd> con le credenziali reali che stai utilizzando nella tua installazione di MySQL.

```
importa MySQLdb come mysql
conn = mysql.connect(user=<adminUser>, password=<adminPwd>,
host='127.0.0.1')
stampa (collegamento)
conn.close()
```

Se l'esecuzione del codice precedente risulta nel seguente output stampato sulla console, allora siamo a posto:

MySQL_connect.py



```
Console Bookmarks
<terminated> C:\Eclipse_NEON_workspace\2nd Edition Python GUI Programming Cookbook\Ch07_Code\MySQL_connect.py
<_mysql.connection open to '127.0.0.1' at 605769b8>
```

Se non riesci a connetterti al server MySQL tramite il *Client da riga di comando* o il pitone *mysqlclient*, allora probabilmente qualcosa è andato storto durante l'installazione. In questo caso, prova a disinstallare, riavviare il PC e quindi eseguire nuovamente l'installazione.

Come funziona...

Per connettere la nostra GUI a un server MySQL, dobbiamo essere in grado di connetterci al server con privilegi amministrativi se vogliamo creare il nostro database. Se il database esiste già, abbiamo solo bisogno dei diritti di autorizzazione per connettere, inserire, aggiornare ed eliminare i dati. Creeremo un nuovo database su un server MySQL nella prossima ricetta.

Configurazione della connessione al database MySQL

Nella ricetta precedente, abbiamo utilizzato il modo più breve per connetterci a un server MySQL inserendo le credenziali richieste per l'autenticazione nel connessione metodo. Sebbene questo sia un approccio rapido per lo sviluppo iniziale, non vogliamo assolutamente esporre le nostre credenziali del server MySQL a nessuno. Invece, noi concedere autorizzazione per accedere a database, tabelle, viste e relativi comandi di database a utenti specifici.

Un modo molto più sicuro per ottenere l'autenticazione da un server MySQL è memorizzare le credenziali in un file di configurazione, che è ciò che faremo in questa ricetta. Useremo il nostro file di configurazione per connetterci al server MySQL e quindi creare il nostro database sul server MySQL.

Useremo questo database in tutte le seguenti ricette.



Prepararsi

Per eseguire il codice mostrato in questa ricetta è necessario l'accesso a un server MySQL in esecuzione con privilegi di amministratore.



La ricetta precedente mostra come installare la Community Edition gratuita di MySQL Server. I privilegi di amministratore ti permetteranno di implementare questa ricetta.

Come farlo...

Innanzitutto, creiamo un dizionario nello stesso modulo del `GUI_MySQL_class.py` codice:

```
# crea un dizionario per contenere le informazioni sulla connessione
dbConfig = {
    'utente': <nomeamministratore>,          # usa il tuo nome amministratore
    'password': <adminPwd>,                  # non è la vera password
    'host': '127.0.0.1',                      # indirizzo IP di localhost
}
```

Successivamente, nel metodo di connessione, decomprimiamo i valori del dizionario. Dai un'occhiata al seguente frammento di codice:

```
mysql.connect('user': <adminName>, 'password': <adminPwd>, 'host': '127.0.0.1')
```

Invece di usare lo snippet precedente, usiamo (`**dbConfig`), che ottiene lo stesso risultato del precedente ma è molto più breve:

```
importa MySQLdb come mysql
# scompatta le credenziali del dizionario
conn = mysql.connect(**dbConfig)
print(conn)
```

Ciò si traduce nella stessa connessione riuscita al server MySQL, ma la differenza è che il metodo di connessione non espone più alcuna informazione mission-critical:



Un server di database è fondamentale per la tua missione. Te ne rendi conto una volta che hai perso i tuoi preziosi dati... e non riesci a trovare alcun backup recente!

MySQL_connect_with_dict.py

```
Console Bookmarks
<terminated> C:\Eclipse_NEON_workspace\2nd Edition Python GUI Programming Cookbook\Ch07_Code\MySQL_connect_with_dict.py
<_mysql.connection open to '127.0.0.1' at 62f229a8>
```

Ora, inserire lo stesso nome utente, password, database e così via in un dizionario nello stesso modulo Python non elimina il rischio che le credenziali vengano visualizzate da chiunque stia esaminando il codice.

Per aumentare la sicurezza del database, spostiamo prima il dizionario nel proprio modulo Python. Chiamiamo il nuovo modulo PythonGuiDBConfig.py.

Quindi importiamo questo modulo e decomprimiamo le credenziali come abbiamo fatto prima:

```
importa GuiDBConfig come guiConf
# scompatta le credenziali del dizionario
conn = mysql.connect(**guiConf.dbConfig) print(conn)
```



Una volta posizionato questo modulo in un luogo sicuro, separato dal resto del codice, abbiamo raggiunto un livello di sicurezza migliore per i nostri dati MySQL.

Ora che sappiamo come connetterci a MySQL e abbiamo i privilegi di amministratore, possiamo creare il nostro database emettendo i seguenti comandi:

```
GUIDB = 'Guida DB'

# scompatta le credenziali del dizionario
conn = mysql.connect(**guiConf.dbConfig)

cursore = conn.cursor()

provare:
    cursor.execute("CREA DATABASE {}"
                   "SET DI CARATTERI PREDEFINITO 'utf8''.format(GUIDB))
```

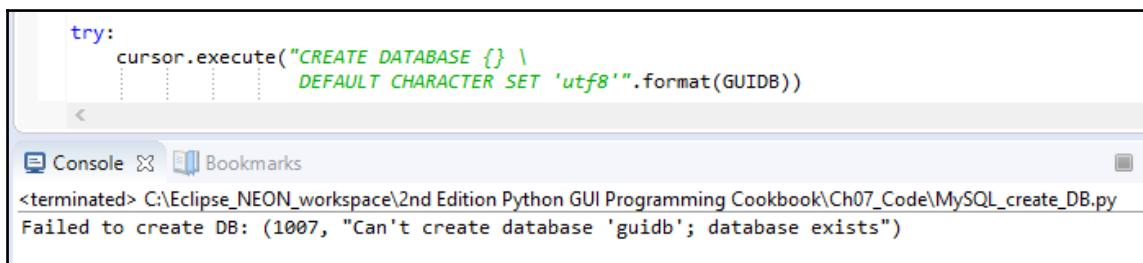
```
tranne mysql.Error come err:  
    print("Impossibile creare DB: {}".format(err))  
  
conn.close()
```

Per eseguire comandi su MySQL, creiamo un oggetto cursore dall'oggetto connessione.

Un cursore è solitamente un posto in una riga specifica in una tabella di database, che spostiamo su o giù nella tabella, ma qui lo usiamo per creare il database stesso. Avvolgiamo il codice Python in a prova...tranne bloccare e utilizzare i codici di errore integrati di MySQL per dirci se qualcosa è andato storto.

Possiamo verificare che questo blocco funzioni eseguendo due volte il codice di creazione del database. La prima volta creerà un nuovo database in MySQL e la seconda stamperà un messaggio di errore che indica che questo database esiste già:

MySQL_create_DB.py



```
try:  
    cursor.execute("CREATE DATABASE {} \\  
    DEFAULT CHARACTER SET 'utf8'".format(GUIDB))  
    ...  
    ...  
  
Console Bookmarks  
<terminated> C:\Eclipse_NEON_workspace\2nd Edition Python GUI Programming Cookbook\Ch07_Code\MySQL_create_DB.py  
Failed to create DB: (1007, "Can't create database 'guidb'; database exists")
```

Possiamo verificare quali database esistono eseguendo il seguente comando MySQL utilizzando la stessa sintassi dell'oggetto cursore. Invece di emettere il CREA DATABASE comando creiamo un cursore e lo usiamo per eseguire il MOSTRA BANCHE DATI comando, il cui risultato recuperiamo e stampiamo sull'output della console:

```
import MySQLdb as mysql import  
GuiDBConfig as guiConf  
  
# scompatta le credenziali del dizionario  
conn = mysql.connect(**guiConf.dbConfig)  
  
cursore = conn.cursor()  
  
cursor.execute("MOSTRA DATABASE")  
print(cursor.fetchall())  
  
conn.close()
```



Recuperiamo i risultati chiamando il metodo sull'oggetto cursore.

L'esecuzione di questo codice ci mostra quali database esistono attualmente nella nostra istanza del server MySQL. Come possiamo vedere dall'output, MySQL viene fornito con diversi database integrati come schema_informazioni, e così via. Abbiamo creato con successo il nostro guida database, che viene mostrato nell'output. Tutti gli altri database illustrati vengono forniti con MySQL:

MySQL_show_DBs.py

A screenshot of a terminal window titled 'Console'. The window shows the command 'C:\Eclipse_NEON_workspace\2nd Edition Python GUI Programming Cookbook\Ch07_Code\MySQL_show_DBs.py' being run. The output lists several MySQL databases: ('information_schema',), ('guidb',), ('mysql',), ('performance_schema',), ('sakila',), ('sys',), and ('world',).

```
<terminated> C:\Eclipse_NEON_workspace\2nd Edition Python GUI Programming Cookbook\Ch07_Code\MySQL_show_DBs.py
('information_schema',), ('guidb',), ('mysql',), ('performance_schema',), ('sakila',), ('sys',), ('world',))
```

Nota come, anche se abbiamo specificato il database quando lo abbiamo creato in lettere maiuscole come GuiDB, il MOSTRA BANCHE DATI comando mostra tutti i database esistenti in MySQL in minuscolo e mostra il nostro database come guidb.

Come funziona...

Per connettere la nostra GUI Python a un database MySQL, dobbiamo prima sapere come connetterci al server MySQL. Ciò richiede la creazione di una connessione e questa connessione sarà accettata da MySQL solo se siamo in grado di fornire le credenziali richieste.

Sebbene sia facile inserire stringhe in una riga di codice Python, quando abbiamo a che fare con i database, dobbiamo essere molto attenti perché l'ambiente di sviluppo sandbox personale di oggi potrebbe facilmente finire per essere accessibile sul World Wide Web entro domani.

Non vuoi compromettere la sicurezza del database e la prima parte di questa ricetta ha mostrato modi per essere più sicuri inserendo le credenziali di connessione al server MySQL in un file separato e, posizionando questo file in una posizione in cui non è accessibile da il mondo esterno, il nostro sistema di database diventerà più sicuro.

In un ambiente di produzione reale, l'installazione del server MySQL, le credenziali di connessione e questo dbConfig file verrebbe gestito dagli amministratori di sistema IT che ti consentirebbero di importare il dbConfig file per connettersi al server MySQL senza che tu sappia quali sono le credenziali effettive. DisimballaggiadbConfig non esporrebbe le credenziali come fa nel nostro codice.

La seconda parte ha creato il nostro database in un'istanza del server MySQL e estenderemo e utilizzeremo questo database nelle seguenti ricette, combinandolo con la nostra GUI Python.

Progettare il database della GUI Python

Prima di iniziare a creare tabelle e inserire dati in esse, dobbiamo progettare il database. A differenza della modifica dei nomi delle variabili Python locali, la modifica di uno schema di database una volta che è stato creato e caricato con i dati non è così facile.

dovremmo FAR CADERE la tabella, il che significa che perderemmo tutti i dati presenti nella tabella. Quindi, prima di eliminare una tabella, dovremmo estrarre i dati, quindi FAR CADERE la tabella, ricrearla e infine reimportare i dati originali.

Hai la foto...

Progettare il nostro database GUI MySQL significa prima pensare a cosa vogliamo che la nostra applicazione Python faccia con esso e poi scegliere i nomi per le nostre tabelle che corrispondono allo scopo previsto.

Prepararsi

Stiamo lavorando con il database MySQL che abbiamo creato nella ricetta precedente, *Configurazione della connessione al database MySQL*. È necessaria un'istanza di MySQL in esecuzione e le due ricette precedenti mostrano come installare MySQL, tutti i driver aggiuntivi necessari e come creare il database che stiamo usando in questo capitolo.

Come farlo...

Innanzitutto, spostiamo i widget dalla nostra GUI Python tra le due schede che abbiamo creato nelle ricette precedenti per organizzare meglio la nostra GUI Python per connettersi a un database MySQL.

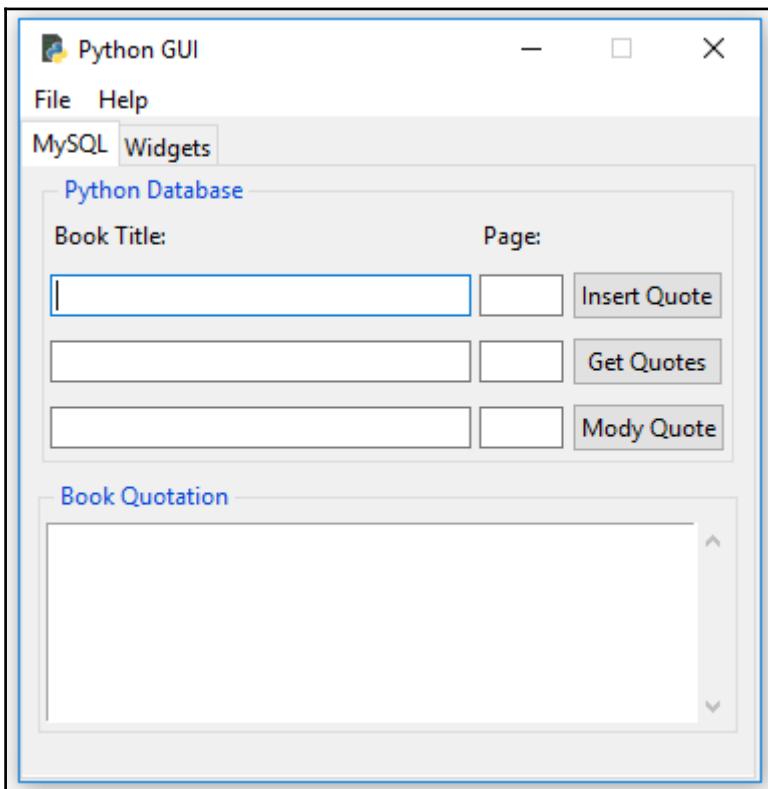
Rinominiamo diversi widget e separiamo il codice che accede ai dati MySQL in quello che era chiamato Tab 1, e sposteremo i widget non correlati a quello che abbiamo chiamato Tab 2 nelle ricette precedenti. Adeguiamo anche alcuni nomi di variabili Python interne per comprendere meglio il nostro codice.



La leggibilità del codice è una virtù del codice e non una perdita di tempo.

La nostra GUI Python rifattorizzata ora ha l'aspetto della seguente schermata. Abbiamo rinominato la prima scheda come **MySQL** e ho creato due tkinter EtichettaFrame widget. Abbiamo etichettato quello sul database Python in alto e contiene due etichette e sei widget di immissione tkinter più tre pulsanti, che abbiamo allineato in quattro righe e tre colonne utilizzando il gestore di layout della griglia tkinter. Inseriremo i titoli e le pagine dei libri nei widget di immissione e facendo clic sui pulsanti si inseriranno, recupereranno o modificheranno le citazioni dei libri. Il EtichettaFrame in basso ha un'etichetta di **Preventivo del libro** e il Testo scorrevole widget che fa parte di questo frame mostrerà i nostri libri e citazioni:

GUI_SQL.py



Creeremo due tabelle SQL per contenere i nostri dati. La prima conterrà i dati per il titolo del libro e la pagina del libro, quindi ci uniremo alla seconda tabella, che conterrà la citazione del libro. Collegheremo insieme le due tabelle tramite le relazioni primarie con le chiavi esterne.

Quindi, creiamo ora la prima tabella del database. Prima di farlo, verifichiamo prima che il nostro database, in effetti, non abbia tabelle. Secondo la documentazione online di MySQL, il comando per visualizzare le tabelle esistenti in un database è il seguente:



14.7.5.37 MOSTRA TABELLE Sintassi

```
MOSTRA [FULL] TABELLE [{FROM | IN} db_name]
[MI PIACE 'schema' | DOVE espr]
```

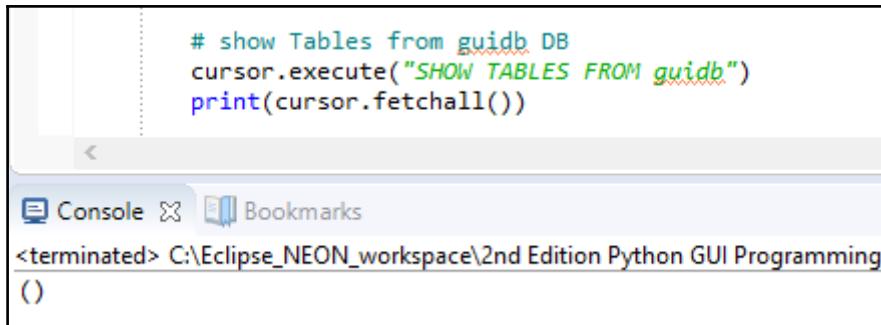
È importante notare che, nella sintassi precedente, gli argomenti tra parentesi quadre, come PIENO, sono opzionali mentre gli argomenti tra parentesi graffe, come, A PARTIRE DAL, sono necessari per il MOSTRA TABELLE comando. Il simbolo della pipa tra A PARTIRE DAL e NEL significa che la sintassi MySQL richiede l'uno o l'altro:

```
# scompatta le credenziali del dizionario
conn = mysql.connect(**guiConf.dbConfig)
# crea il cursore
cursore = conn.cursor()
# esegui il comando
cursor.execute("MOSTRA TABELLE DA guidb")
print(cursor.fetchall())

# chiude la connessione a MySQL
conn.close()
```

Quando eseguiamo il comando SQL in Python, otteniamo il risultato atteso, che è una tupla vuota che ci mostra che il nostro database attualmente non ha tabelle:

GUI_SQL_class.py



The screenshot shows an Eclipse IDE interface with a Python script in the editor and its execution results in the console.

```
# show Tables from guidb DB
cursor.execute("SHOW TABLES FROM guidb")
print(cursor.fetchall())
```

Console Bookmarks

<terminated> C:\Eclipse_NEON_workspace\2nd Edition Python GUI Programming
()

Possiamo anche selezionare prima il database eseguendo il comando USA <DB> comando e quindi non dobbiamo passarlo nel MOSTRA TABELLE comando perché abbiamo già selezionato il database con cui vogliamo parlare. Il codice seguente crea lo stesso vero risultato del precedente:

```
cursor.execute("USA guidb")
cursor.execute("MOSTRA TABELLE")
```

Ora che sappiamo come verificare che il nostro database non abbia tabelle, creiamone alcune. Dopo aver creato due tabelle, verificheremo che siano veramente entrate nel nostro database utilizzando gli stessi comandi di prima.

Creiamo la prima tabella, denominata libri, eseguendo il seguente codice:

```
# connettersi scompattando le credenziali del dizionario
conn = mysql.connect(**guiConf.dbConfig)

# crea il cursore
cursore = conn.cursor()

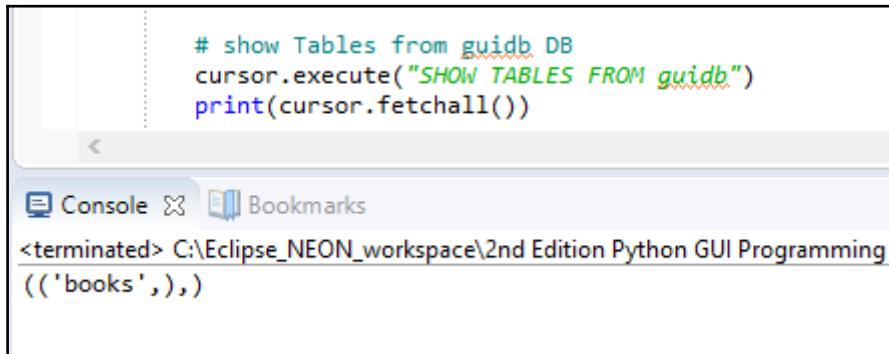
# seleziona DB
cursor.execute("USA guidb")

# crea una tabella all'interno del DB
cursor.execute("CREATE TABLE Libri (
    Book_ID INT NOT NULL AUTO_INCREMENT,
    Book_Title VARCHAR(25) NOT NULL, Book_Page
    INT NOT NULL,
    CHIAVE PRIMARIA (Libro_ID)
) MOTORE=InnoDB")

# chiude la connessione a MySQL
conn.close()
```

Possiamo verificare che la tabella sia stata creata nel nostro database eseguendo i seguenti comandi:

GUI_MySQL_class.py



The screenshot shows an Eclipse IDE interface with a code editor and a terminal window. The code editor contains Python code to show tables in a database:

```
# show Tables from guidb DB
cursor.execute("SHOW TABLES FROM guidb")
print(cursor.fetchall())
```

The terminal window shows the output of the command:

```
<terminated> C:\Eclipse_NEON_workspace\2nd Edition Python GUI Programming
(('books',),)
```

Ora il risultato non è più una tupla vuota ma una tupla che contiene una tupla, mostrando il libri tabella che abbiamo appena creato.

Possiamo usare il client da riga di comando MySQL per vedere le colonne nella nostra tabella. Per fare ciò, dobbiamo accedere come utente root. Dobbiamo anche aggiungere un punto e virgola alla fine del comando.



Su Windows, è sufficiente fare doppio clic sul collegamento del client dalla riga di comando MySQL, che viene installato automaticamente durante l'installazione di MySQL.

Se non hai un collegamento sul desktop, puoi trovare l'eseguibile nel seguente percorso per una tipica installazione predefinita:

C:\Programmi\MySQL\MySQL Server 5.7\bin\mysql.exe

Senza una scorciatoia per eseguire il client MySQL, devi passargli alcuni parametri:

- C:\Programmi\MySQL\MySQL Server 5.7\bin\mysql.exe
- - tu root
- - p

Facendo doppio clic sul collegamento o utilizzando la riga di comando con il percorso completo dell'eseguibile e passando i parametri richiesti, verrà visualizzato il client della riga di comando MySQL, che richiede di inserire la password per l'utente root:

```
cmd Command Prompt - "C:\Program Files\MySQL\MySQL Server 5.7\bin\mysql.exe" -u root -p

C:\>"C:\Program Files\MySQL\MySQL Server 5.7\bin\mysql.exe" -u root -p
Enter password: *****
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 35
Server version: 5.7.17-log MySQL Community Server (GPL)

Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

Se ricordi la password che hai assegnato all'utente root durante l'installazione, puoi eseguire il MOSTRA COLONNE DA libri; comando, come mostrato nella schermata seguente. Questo mostrerà le colonne della nostrolibri tabella dalla nostra guidb:

```
mysql> USE guidb
Database changed
mysql> SHOW COLUMNS FROM books;
+-----+-----+-----+-----+-----+-----+
| Field | Type   | Null | Key  | Default | Extra       |
+-----+-----+-----+-----+-----+-----+
| Book_ID | int(11) | NO   | PRI  | NULL    | auto_increment |
| Book_Title | varchar(25) | NO   |      | NULL    |             |
| Book_Page | int(11) | NO   |      | NULL    |             |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.11 sec)

mysql>
```



Quando si eseguono comandi nel client MySQL, la sintassi non è Python.

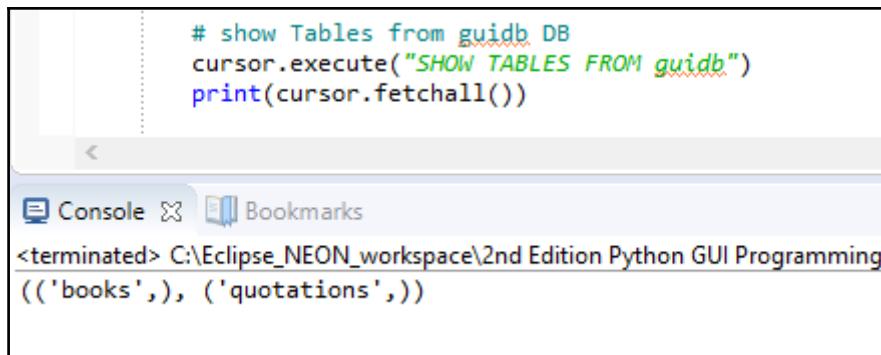
Successivamente, creeremo la seconda tabella, che conserverà le citazioni del libro e del diario. Lo creeremo eseguendo il seguente codice:

```
# seleziona DB
cursor.execute("USA guidb")

# crea la seconda tabella all'interno del DB
cursor.execute("CREATE TABLE Quotazioni (
    Quota_ID INT,
    Preventivo VARCHAR(250),
    Books_Book_ID INT,
    CHIAVE ESTERA (Libri_Libro_ID)
        RIFERIMENTI Libri (ID_Libro) SU
        ELIMINA CASCADE
) MOTORE=InnoDB")
```

L'esecuzione del MOSTRA TABELLE Il comando ora mostra che il nostro database ha due tabelle:

GUI_SQL_class.py



```
# show Tables from guidb DB
cursor.execute("SHOW TABLES FROM guidb")
print(cursor.fetchall())
```

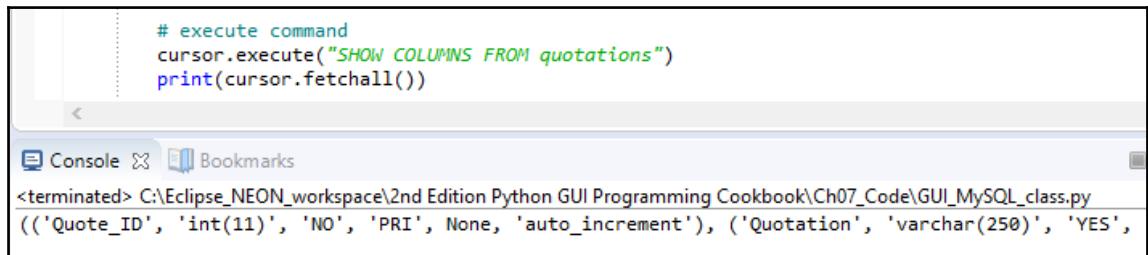
Console Bookmarks

<terminated> C:\Eclipse_NEON_workspace\2nd Edition Python GUI Programming

(('books',), ('quotations',))

Possiamo vedere le colonne eseguendo il comando SQL usando Python:

GUI_SQL_class.py



```
# execute command
cursor.execute("SHOW COLUMNS FROM quotations")
print(cursor.fetchall())
```

Console Bookmarks

<terminated> C:\Eclipse_NEON_workspace\2nd Edition Python GUI Programming Cookbook\Ch07_Code\GUI_SQL_class.py

(('Quote_ID', 'int(11)', 'NO', 'PRI', None, 'auto_increment'), ('Quotation', 'varchar(250)', 'YES',

L'utilizzo del client MySQL potrebbe presentare i dati in un formato migliore. Potremmo anche usare la bella stampa di Python (stampa) caratteristica:

GUI_MySQL_class.py

The screenshot shows an Eclipse IDE interface. In the top editor pane, there is a Python script with the following code:

```
from pprint import pprint
# execute command
cursor.execute("SHOW COLUMNS FROM quotations")
pprint(cursor.fetchall())
```

In the bottom console pane, the output of the script is shown:

```
<terminated> C:\Eclipse_NEON_workspace\2nd Edition Python GUI Programming Cookbook
(['Quote_ID', 'int(11)', 'NO', 'PRI', None, 'auto_increment'],
 ['Quotation', 'varchar(250)', 'YES', '', None, ''],
 ['Books_Book_ID', 'int(11)', 'YES', 'MUL', None, ''])
```

Il client MySQL mostra ancora le nostre colonne in un formato più chiaro, che può essere visto quando si esegue questo client.

Come funziona...

Abbiamo progettato il nostro database GUI Python e rifactorizzato la nostra GUI in preparazione all'utilizzo del nostro nuovo database. Abbiamo quindi creato un database MySQL e creato due tabelle al suo interno.

Abbiamo verificato che le tabelle siano state inserite nel nostro database utilizzando sia Python che il client MySQL fornito con il server MySQL.

Nella prossima ricetta, inseriremo i dati nelle nostre tabelle.

Utilizzo del comando SQL INSERT

Questa ricetta presenta l'intero codice Python che mostra come creare ed eliminare database e tabelle MySQL e come visualizzare database, tabelle, colonne e dati esistenti della nostra istanza MySQL.

Dopo aver creato il database e le tabelle, inseriremo i dati nelle due tabelle che stiamo creando in questa ricetta.



Utilizziamo una relazione tra chiave primaria e chiave esterna per collegare i dati delle due tabelle.

Entreremo nei dettagli di come funziona nelle seguenti due ricette, dove modifichiamo ed eliminiamo i dati nel nostro database MySQL.

Prepararsi

Questa ricetta si basa sul database MySQL che abbiamo creato nella ricetta precedente, *Progettare il database della GUI Python*, e mostra anche come eliminare e ricreare il GuiDB.



L'eliminazione del database, ovviamente, elimina tutti i dati che il database aveva nelle sue tabelle, quindi ti mostreremo come reinserirle anche quei dati.

Come farlo...

L'intero codice del nostro GUI_MySQL_class.py è presente nella cartella del codice di questo capitolo, che è disponibile per il download da <https://github.com/PacktPublishing/Python-GUI-Programmazione-Ricettario-Seconda-Eduzione>. Crea il database, aggiunge tabelle a it, quindi inserisce i dati nelle due tabelle che abbiamo creato.

Qui, delineeremo il codice senza mostrare tutti i dettagli di implementazione per preservare lo spazio perché ci vorrebbero troppe pagine per mostrare l'intero codice:

```
import MySQLdb as mysql
import Ch07_Code.GuiDBConfig as guiConf

class MySQL():
    # variabile di classe
    GUIDB = 'Guida DB'
    #
    def connettersi (auto):
        # connettersi scompattando le credenziali del dizionario
        conn = mysql.connector.connect(**guiConf.dbConfig)
        # crea il cursore
        cursore = conn.cursor()
        ritorno conn, cursore
    #

```

```
def close(self, cursore, conn):
    # chiudi il cursore
#
# -----
def showDBs(self):
    # connettersi a MySQL
#
# -----
def createGuiDB(self):
    # connettersi a MySQL
#
# -----
def dropGuiDB(self):
    # connettersi a MySQL
#
# -----
def useGuiDB(self, cursore):
    """Prevede una connessione aperta."""
    # seleziona DB
#
# -----
def createTables(self):
    # connettersi a MySQL
    # crea una tabella all'interno del DB
#
# -----
def dropTables(self):
    # connettersi a MySQL
#
# -----
def showTables(self):
    # connettersi a MySQL
#
# -----
def insertBooks(self, title, page, bookQuote):
    # connettersi a MySQL
    # inserire dati
#
# -----
def insertBooksExample(self):
    # connettersi a MySQL
    # inserire dati hardcoded
#
# -----
def showBooks(self):
    # connettersi a MySQL
#
# -----
def showColumns(self):
    # connettersi a MySQL
#
# -----
def showData(self):
    # connettersi a MySQL
#
# -----
if __name__ == '__main__':
    # Crea istanza di classe
    mySQL = MySQL()
```

L'esecuzione del codice precedente crea le seguenti tabelle e dati nel database che abbiamo creato:

```
mysql> USE guidb
Database changed
mysql> SELECT * FROM books;
+-----+-----+
| Book_ID | Book_Title      | Book_Page |
+-----+-----+
| 1       | Design Patterns   |      7    |
| 2       | xUnit Test Patterns |     31    |
+-----+-----+
2 rows in set (0.10 sec)

mysql> SELECT * FROM quotations;
+-----+-----+
| Quote_ID | Quotation          | Books_Book_ID |
+-----+-----+
| 1         | Programming to an Interface, not an Implementation | 1 |
| 2         | Philosophy of Test Automation | 2 |
+-----+-----+
2 rows in set (0.00 sec)

mysql>
```

Come funziona...

Abbiamo creato un database MySQL, ci siamo collegati e quindi abbiamo creato due tabelle che contengono i dati per una citazione di un libro o di un diario preferito.

Abbiamo distribuito i dati tra due tabelle perché le citazioni tendono ad essere piuttosto grandi mentre i titoli dei libri e i numeri delle pagine dei libri sono molto brevi. In questo modo, possiamo aumentare l'efficienza del nostro database.



Nel linguaggio del database SQL, la separazione dei dati in tabelle separate è chiamata normalizzazione.

Utilizzo del comando SQL UPDATE

Questa ricetta utilizzerà il codice della ricetta precedente, *Utilizzando il comando SQL INSERT*, spiegalo in modo più dettagliato, quindi estendi il codice per aggiornare i nostri dati.

Per aggiornare i dati che abbiamo precedentemente inserito nelle nostre tabelle del database MySQL, utilizziamo SQL AGGIORNARE comando.

Prepararsi

Questa ricetta si basa sulla ricetta precedente, *Utilizzando il comando SQL INSERT*, quindi leggi e studia la ricetta precedente per seguire la codifica in questa ricetta, dove modifichiamo i dati esistenti.

Come farlo...

Innanzitutto, mostreremo i dati da modificare eseguendo il seguente comando da Python a MySQL:

```
importa MySQLdb come mysql
import Ch07_Code.GuiDBConfig come guiConf

classe MySQL():
    # variabile di classe
    GUIDB = 'Guida DB'
    # -----
    def showData(self):
        # connettersi a MySQL
        conn, cursore = self.connect()

        self.useGuiDB(cursore)

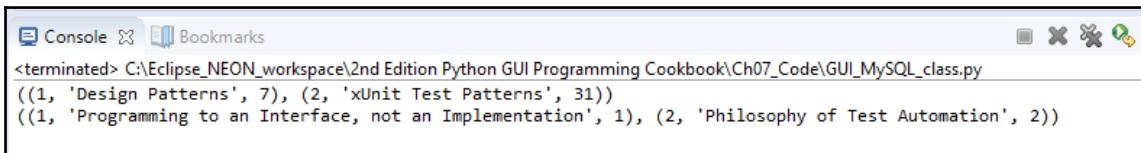
        # esegui il comando
        cursor.execute("SELECT * FROM libri")
        print(cursor.fetchall())

        cursor.execute("SELECT * FROM citazioni")
        print(cursor.fetchall())

        # chiudi cursore e connessione
        self.close(cursore, conn)
# ===== if
__name__ == '__main__':
    # Crea istanza di classe
    mySQL = MySQL()
    mySQL.showData()
```

L'esecuzione del codice produce il seguente risultato:

GUI_SQL_class.py



```
<terminated> C:\Eclipse_NEON_workspace\2nd Edition Python GUI Programming Cookbook\Ch07_Code\GUI_SQL_class.py
((1, 'Design Patterns', 7), (2, 'xUnit Test Patterns', 31))
((1, 'Programming to an Interface, not an Implementation', 1), (2, 'Philosophy of Test Automation', 2))
```

Potremmo non essere d'accordo con la Gang of Four, quindi cambiamo la loro famosa citazione di programmazione.



The Gang of Four sono i quattro autori che hanno creato il famoso libro chiamato *Modelli di progettazione*, che ha fortemente influenzato la nostra intera industria del software a riconoscere, pensare e codificare utilizzando modelli di progettazione del software.

Lo faremo aggiornando il nostro database delle citazioni preferite. Innanzitutto, recuperiamo il valore della chiave primaria cercando il titolo del libro e poi passiamo quel valore alla nostra ricerca della citazione:

```
# -----
def updateGOF(self):
    # connettersi a MySQL
    conn, cursore = self.connect()

    self.useGuiDB(cursore)

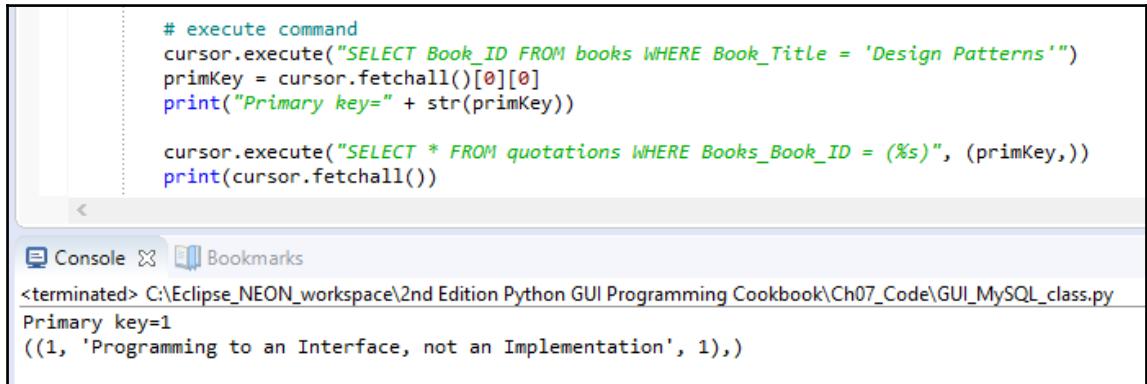
    # esegui il comando
    cursor.execute("SELECT Book_ID FROM libri WHERE Book_Title =
                   'Modelli di progettazione'")
    primKey = cursor.fetchall()[0][0] print("Chiave
    primaria=" + str(PrimKey))

    cursor.execute("SELECT * FROM citazioni WHERE Books_Book_ID =
                   (%s)", (primKey,))
    print(cursore.fetchall())

    # chiudi cursore e connessione
    self.close(cursore, conn)
# ===== if
__name__ == '__main__':
    mySQL = MySQL()           # Crea istanza di classe
    mySQL.updateGOF()
```

Questo ci dà il seguente risultato:

GUI_SQL_class.py



The screenshot shows the Eclipse IDE interface with a code editor and a terminal window. The code in the editor is a Python script named `GUI_SQL_class.py`. It contains SQL queries to select primary keys and quotation details from MySQL tables. The terminal window shows the script has run successfully, printing the primary key value `1` and a tuple of quotation details.

```
# execute command
cursor.execute("SELECT Book_ID FROM books WHERE Book_Title = 'Design Patterns'")
primKey = cursor.fetchall()[0][0]
print("Primary key=" + str(primKey))

cursor.execute("SELECT * FROM quotations WHERE Books_Book_ID = (%s)", (primKey,))
print(cursor.fetchall())
```

Console Bookmarks

<terminated> C:\Eclipse_NEON_workspace\2nd Edition Python GUI Programming Cookbook\Ch07_Code\GUI_SQL_class.py

Primary key=1
((1, 'Programming to an Interface, not an Implementation', 1),)

Ora che conosciamo la chiave primaria della citazione, possiamo aggiornare la citazione eseguendo i seguenti comandi:

```
# -----
def showDataWithReturn(self):
    # connettersi a MySQL
    conn, cursore = self.connect()

    self.useGuiDB(cursore)

    # esegui il comando
    cursor.execute("SELECT Book_ID FROM libri WHERE Book_Title =
                   'Modelli di progettazione'")
    primKey = cursor.fetchall()[0][0]
    print(primKey)

    cursor.execute("SELECT * FROM citazioni WHERE Books_Book_ID =
                   (%s)", (primKey,))
    print(cursore.fetchall())

    cursor.execute("AGGIORNA citazioni SET Quotazione =
                   (%s) WHERE Books_Book_ID = (%s)",
                   ("Pythonic Duck Typing: se cammina come un'anatra e parla come
                   un'anatra, probabilmente è un'anatra...", primKey))

    # commit transazione
    conn.commit()
```

```
cursor.execute("SELECT * FROM citazioni WHERE Books_Book_ID =  
                (%s)", (primaryKey,))  
print(cursore.fetchall())  
  
# chiudi cursore e connessione  
self.close(cursore, conn)  
  
# ===== if  
_name_ == '__main__':  
    # Crea istanza di classe  
    mySQL = MySQL()  
    # -----  
    mySQL.updateGOF()  
    libro, citazione = mySQL.showDataWithReturn()  
    print(libro, citazione)
```

Eseguendo il codice precedente, rendiamo questo classico della programmazione più Pythonico.

Come si può vedere nello screenshot seguente, prima di eseguire il codice precedente, il nostro titolo con Book_ID 1 era collegato tramite una relazione da primaria a chiave esterna alla citazione nel Libri_ID_Libro colonna della tabella delle quotazioni. Questa è la citazione originale di

Modelli di progettazione libro.

Abbiamo quindi aggiornato la quotazione relativa a questo ID tramite SQL AGGIORNARE comando.

Nessuno degli ID è cambiato ma la citazione che ora è associata con Book_ID 1 è cambiato come si può vedere nella seconda finestra del client MySQL:

```
mysql> USE guidb
Database changed
mysql> SELECT * FROM books;
+-----+-----+
| Book_ID | Book_Title      | Book_Page |
+-----+-----+
| 1 | Design Patterns    |      7 |
| 2 | xUnit Test Patterns |     31 |
+-----+
2 rows in set (0.10 sec)

mysql> SELECT * FROM quotations;
+-----+-----+-----+
| Quote_ID | Quotation          | Books_Book_ID |
+-----+-----+-----+
| 1 | Programming to an Interface, not an Implementation | 1 |
| 2 | Philosophy of Test Automation | 2 |
+-----+-----+
```



```
mysql> SELECT * FROM books;
+-----+-----+
| Book_ID | Book_Title      | Book_Page |
+-----+-----+
| 1 | Design Patterns    |      7 |
| 2 | xUnit Test Patterns |     31 |
+-----+
2 rows in set (0.00 sec)

mysql> SELECT * FROM quotations;
+-----+
| Quote_ID | Quotation          |
+-----+
| 1 | Pythonic Duck Typing: If it walks like a duck and talks like a duck it probably is a duck... |
| 2 | Philosophy of Test Automation |
+-----+
```

Come funziona...

In questa ricetta, abbiamo recuperato i dati esistenti dal nostro database e dalle tabelle del database che abbiamo creato nelle ricette precedenti. Abbiamo inserito i dati nelle tabelle e aggiornato i nostri dati utilizzando SQLAGGIORNARE comando.

Utilizzo del comando SQL DELETE

In questa ricetta, useremo SQL ELIMINA comando per cancellare i dati che abbiamo creato nella ricetta precedente, *Utilizzo del comando SQL UPDATE*.

Anche se a prima vista l'eliminazione dei dati può sembrare banale, una volta ottenuto un progetto di database piuttosto grande in produzione, le cose potrebbero non essere più così facili.

Poiché abbiamo progettato il nostro database GUI mettendo in relazione due tabelle tramite una relazione tra chiave primaria e esterna, quando eliminiamo determinati dati, non finiamo con record orfani perché questo design del database si occupa delle eliminazioni a cascata.

Prepararsi

Questa ricetta utilizza il database MySQL, le tabelle, nonché i dati inseriti in tali tabelle dalla ricetta precedente, *Utilizzo del comando SQL UPDATE*. Per dimostrare come creare record orfani, dovremo modificare il design di una delle nostre tabelle del database.

Come farlo...

Abbiamo mantenuto il design del nostro database semplice utilizzando solo due tabelle del database.

Anche se questo funziona quando eliminiamo i dati, c'è sempre la possibilità di finire con record orfani. Ciò significa che eliminiamo i dati in una tabella ma in qualche modo non eliminiamo i dati correlati in un'altra tabella SQL.

Se creiamo il nostro citazioni tabella senza una relazione di chiave esterna con libri tabella, possiamo finire con record orfani:

```
# crea la seconda tabella all'interno del DB --
# Nessuna relazione CHIAVE ESTERA con la tabella Libri
cursor.execute("CREATE TABLE Quotazioni (
    Quote_ID INT AUTO_INCREMENT,
    Preventivo VARCHAR(250),
    Books_Book_ID INT,
    CHIAVE PRIMARIA (Quote_ID)
) MOTORE=InnoDB")
```

Dopo aver inserito i dati nei libri e citazioni tabelle, se eseguiamo a ELIMINA dichiarazione, stiamo solo eliminando il libro con Book_ID 1 mentre la relativa citazione con il Libri_ID_Libro 1 viene lasciato indietro.

Questo è un *record orfano*. Non esiste più un registro contabile che abbia un Book_ID di 1:

```
mysql> SELECT * FROM books;
+-----+-----+
| Book_ID | Book_Title      | Book_Page |
+-----+-----+
|      2  | xUnit Test Patterns |        31 |
+-----+-----+
1 row in set (0.00 sec)

mysql> SELECT * FROM quotations;
+-----+-----+-----+
| Quote_ID | Quotation           | Books_Book_ID |
+-----+-----+-----+
|      1  | Programming to an Interface, not an Implementation |          1 |
|      2  | Philosophy of Test Automation   |          2 |
+-----+-----+-----+
2 rows in set (0.00 sec)

mysql>
```

Questa situazione può creare disordine, che evitiamo utilizzando le eliminazioni a cascata.

Lo facciamo nella creazione delle tabelle aggiungendo alcuni vincoli al database. Quando abbiamo creato la tabella che contiene le citazioni di una ricetta precedente, abbiamo creato il nostro citazioni tabella con un vincolo di chiave esterna che fa esplicito riferimento alla chiave primaria della tabella libri, collegando i due:

```
# crea la seconda tabella all'interno del DB
cursor.execute("CREATE TABLE Quotazioni (
    Quote_ID INT AUTO_INCREMENT,
    Preventivo VARCHAR(250),
    Books_Book_ID INT,
    CHIAVE PRIMARIA (Quote_ID),
    CHIAVE ESTERA (Libri.Libro_ID)
        RIFERIMENTI Libri (ID_Libro) SU
        ELIMINA CASCATA
    ) MOTORE=InnoDB")
```

Il CHIAVE ESTERA la relazione include il SU ELIMINA CASCATA attributo, che in pratica dice al nostro server MySQL di eliminare i record correlati in questa tabella quando vengono eliminati i record a cui si riferisce questa chiave esterna.



Senza specificare il SU ELIMINA CASCATA attributo nella creazione della nostra tabella non possiamo né eliminare né aggiornare i nostri dati perché un AGGIORNARE è un ELIMINA seguito da un INSERIRE.

A causa di questo design, nessun record orfano verrà lasciato indietro, che è ciò che vogliamo.



In MySQL, dobbiamo specificare MOTORE=InnoDB su entrambe le tabelle correlate per utilizzare le relazioni tra chiave primaria e esterna.

Mostriamo i dati nel nostro database:

```
# ===== if
__name__ == '__main__':
    # Crea istanza di classe
    mySQL = MySQL()
    mySQL.showData()
```

Questo ci mostra i seguenti dati nelle nostre tabelle del database:

GUI_MySQL_class.py

```
Console Bookmarks
<terminated> C:\Eclipse_NEON_workspace\2nd Edition Python GUI Programming Cookbook\Ch07_Code\GUI_MySQL_class.py
((1, 'Design Patterns', 7), (2, 'xUnit Test Patterns', 31))
((1, 'Programming to an Interface, not an Implementation', 1), (2, 'Philosophy of Test Automation', 2))
```

Questo ci mostra che abbiamo due record che sono correlati tramite relazioni primarie a chiavi esterne.

Quando ora eliminiamo un record nel libri tabella, ci aspettiamo il relativo record nella citazioni tabella da eliminare anche mediante un'eliminazione a cascata. Proviamo questo eseguendo i seguenti comandi SQL in Python:

```
import MySQLdb as mysql
import Ch07_Code.GuiDBConfig as guiConf

classe MySQL():
    # -----
    def deleteRecord(self):
        # connettersi a MySQL
        conn, cursore = self.connect()
```

```
self.useGuiDB(cursore)

# esegui il comando
cursor.execute("SELECT Book_ID FROM libri WHERE Book_Title =
    'Modelli di progettazione'")
primKey = cursor.fetchall()[0][0]
# print(primKey)

cursor.execute("DELETE FROM libri WHERE Book_ID = (%s)",
    (PrimKey,))

# commit transazione
conn.commit()

# chiudi cursore e connessione
self.close(cursore, conn)
# ===== if
__name__ == '__main__':
    # Crea istanza di classe
    mySQL = MySQL()
    # -----
    mySQL.deleteRecord()
    mySQL.showData()
```

Dopo aver eseguito i comandi precedenti per eliminare i record, otteniamo i seguenti nuovi risultati:

GUI_SQL_class.py



```
#-----#
mySQL.deleteRecord()
mySQL.showData()

<terminated> C:\Eclipse_NEON_workspace\2nd Edition Python GUI Programming Cookbook\Ch07_Code\GUI_SQL_class.py
((2, 'xUnit Test Patterns', 31),)
((2, 'Philosophy of Test Automation', 2),)
```



Il famoso Modelli di progettazione sono spariti dal nostro database di citazioni preferite...

Come funziona...

Abbiamo attivato le eliminazioni a cascata in questa ricetta progettando il nostro database in modo solido tramite relazioni da chiave primaria a chiave esterna con eliminazioni a cascata.

Ciò mantiene i nostri dati sani e integri.

Nella prossima ricetta utilizzeremo il codice del nostro MySQL.py modulo dalla nostra GUI Python.

Memorizzazione e recupero dei dati dal nostro database MySQL

Useremo la nostra GUI Python per inserire i dati nelle nostre tabelle del database MySQL. Abbiamo già rifattorizzato la GUI che abbiamo creato nelle ricette precedenti nella nostra preparazione per la connessione e l'utilizzo di un database.

Utilizzeremo due widget di immissione della casella di testo in cui possiamo digitare il titolo del libro o del diario e il numero di pagina. Useremo anche aTesto scorrevole widget in cui digitare le nostre citazioni di libri preferiti, che poi memorizzeremo nel nostro database MySQL.

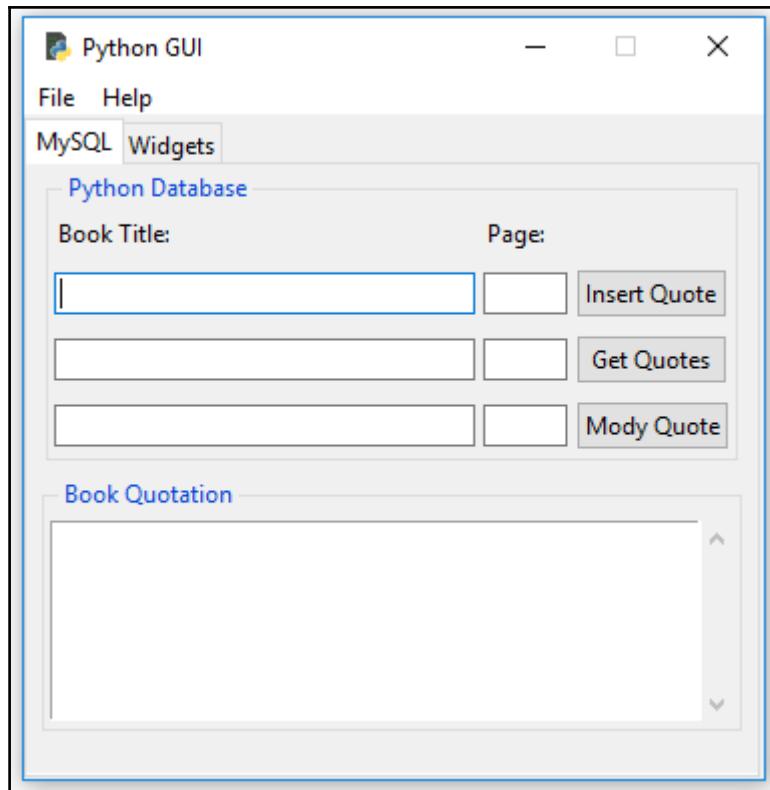
Prepararsi

Questa ricetta si baserà sul database MySQL e sulle tabelle che abbiamo creato nelle ricette precedenti.

Come farlo...

Inseriremo, recupereremo e modificheremo le nostre citazioni preferite utilizzando la nostra GUI Python. Abbiamo rifattorizzato la scheda MySQL della nostra GUI in preparazione per questo:

GUI_SQL.py



Per fare in modo che i pulsanti facciano qualcosa, li collegheremo alle funzioni di callback come abbiamo fatto nelle ricette precedenti. Mostreremo i dati in inTesto scorrevole widget sotto i pulsanti.

Per fare ciò, importeremo il MySQL.py modulo, come abbiamo fatto prima. L'intero codice che comunica con la nostra istanza del server MySQL e il database risiede in questo modulo, che è una forma di encapsulamento del codice nello spirito della programmazione orientata agli oggetti.

Colleghiamo il **Inserisci preventivo** pulsante alla seguente funzione di richiamata:

```
# Aggiunta di un pulsante
self.action = ttk.Button(self.mySQL, text="Inserisci citazione",
                        comando=self.insertQuote)
self.action.grid(colonna=2, riga=1)

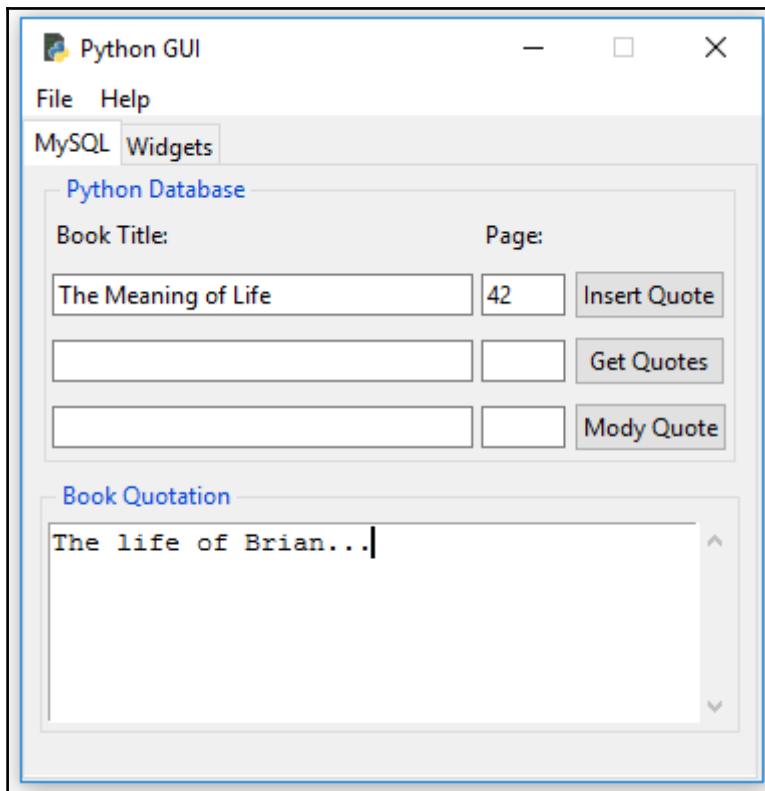
# Pulsante di richiamata
def insertQuote(self):
    titolo = self.bookTitle.get() pagina =
    self.pageNumber.get()
```

```
quote = self.quote.get(1.0, tk.END) print(titolo)
```

```
stampa (citaione)  
self.mySQL.insertBooks(titolo, pagina, citazione)
```

Quando ora eseguiamo il nostro codice, possiamo inserire i dati dalla nostra GUI Python nel nostro database MySQL:

GUI_SQL.py

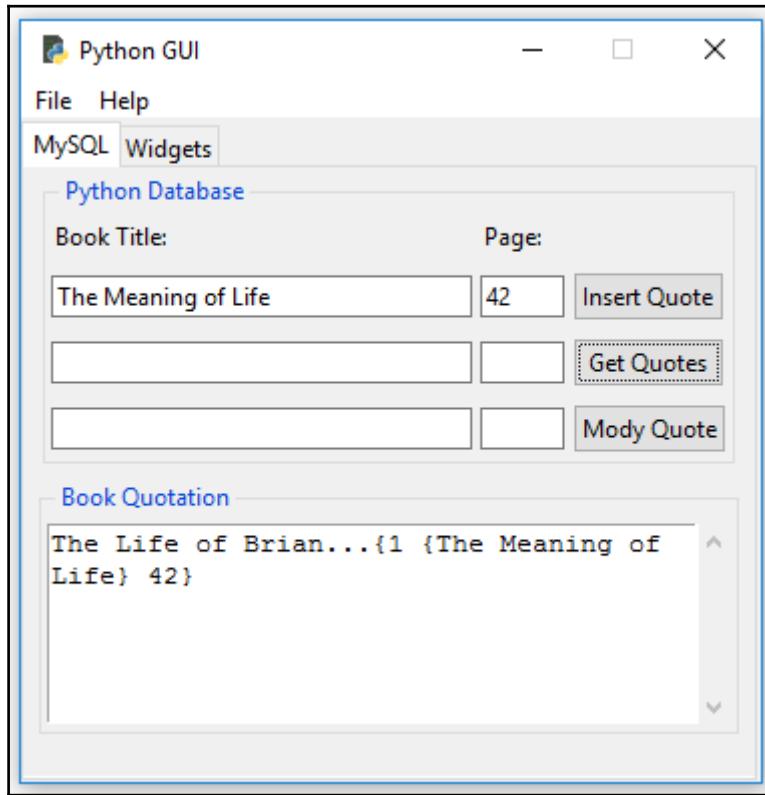


Dopo aver inserito un titolo e una pagina del libro più una citazione dal libro o dal film, inseriamo i dati nel nostro database facendo clic sul pulsante **Inserisci preventivo** pulsante.

Il nostro design attuale consente titoli, pagine e una citazione. Possiamo anche inserire le nostre citazioni preferite dai film. Sebbene un film non abbia pagine, possiamo utilizzare la colonna della pagina per inserire l'ora approssimativa in cui si è verificata la citazione all'interno del film.

Successivamente, possiamo verificare che tutti questi dati siano stati inseriti nelle nostre tabelle del database emettendo gli stessi comandi che abbiamo usato in precedenza:

GUI_SQL.py



Dopo aver inserito i dati, possiamo verificare che siano stati inseriti nelle nostre due tabelle MySQL facendo clic su **Ottenere le quotazioni** pulsante, che mostra quindi i dati che abbiamo inserito nelle nostre due tabelle del database MySQL, come mostrato nello screenshot precedente.

Facendo clic su **Ottenere le quotazioni** button richiama il metodo di callback che abbiamo associato all'evento click del pulsante. Questo ci fornisce i dati che mostriamo nel nostroTesto scorrevole aggeggio:

```
# Aggiunta di un pulsante
self.action1 = ttk.Button(self.mySQL, text="Get Quotes",
                           comando=self.getQuote)
self.action1.grid(column=2, row=2)

# Pulsante di richiamata
```

```
def getQuote(self):
    allBooks = self.mySQL.showBooks()
    print(allBooks)
    self.quote.insert(tk.INSERT, allBooks)
```

Noi usiamo il self.mySQL variabile di istanza di classe per invocare il mostraLibri() metodo, che fa parte del MySQL classe che abbiamo importato:

```
da Ch07_Code.GUI_MySQL_class importa la classe MySQL
OOP():
    def __init__(self):
        # crea un'istanza MySQL
        self.mySQL = MySQL()

classe MySQL():
    # -----
    def showBooks(self):
        # connettersi a MySQL
        conn, cursore = self.connect()

        self.useGuiDB(cursore)

        # risultati di stampa
        cursor.execute("SELECT * FROM Books") allBooks
        = cursor.fetchall()
        stampa(tutti i libri)

        # chiudi cursore e connessione
        self.close(cursore, conn)

    ritorna tuttiLibri
```

Come funziona...

In questa ricetta abbiamo importato il modulo Python che abbiamo scritto che contiene tutta la logica di codifica per connettersi al nostro database MySQL e sa come inserire, aggiornare ed eliminare i dati.

Ora abbiamo collegato la nostra GUI Python a questa logica SQL.

Utilizzo del workbench MySQL

MySQL ha una GUI molto carina che possiamo scaricare gratuitamente. Si chiama **MySQL Workbench**.

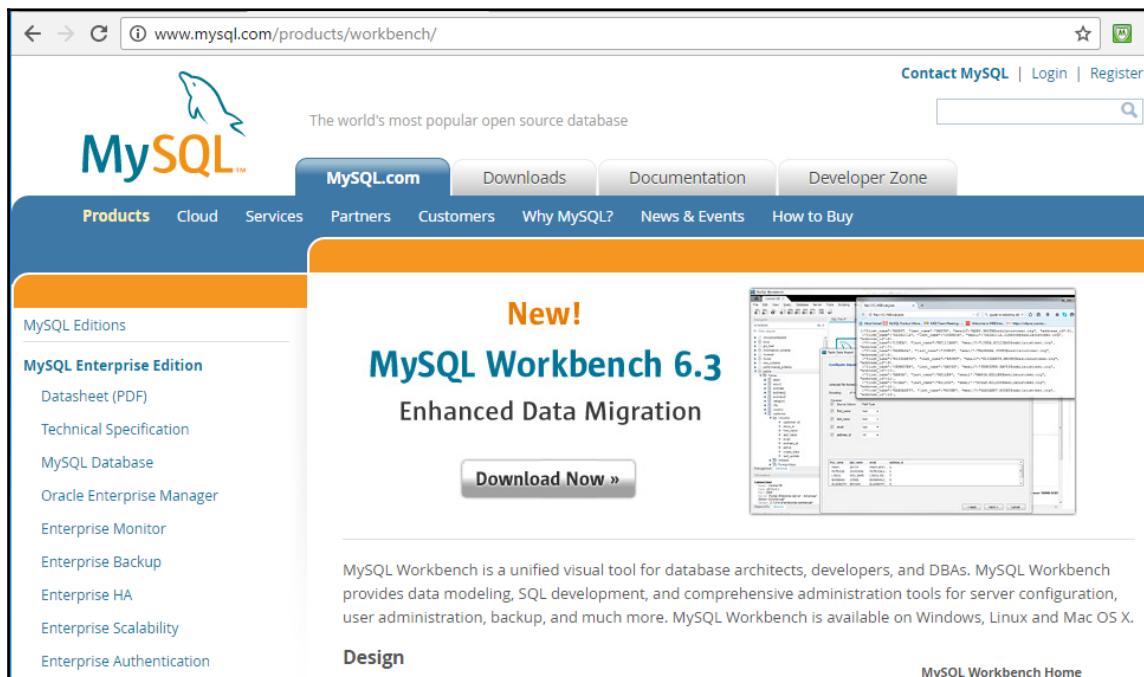
In questa ricetta, installeremo con successo questo workbench e poi lo useremo per eseguire query SQL sul GuiDB che abbiamo creato nelle ricette precedenti, *Configurazione della connessione al database MySQL*, *Progettazione del database GUI Python* e così via.

Prepararsi

Per utilizzare questa ricetta, avrai bisogno del database MySQL che abbiamo sviluppato nelle ricette precedenti. Avrai anche bisogno di un server MySQL in esecuzione.

Come farlo...

Possiamo scaricare MySQL Workbench dal sito Web ufficiale di MySQL:



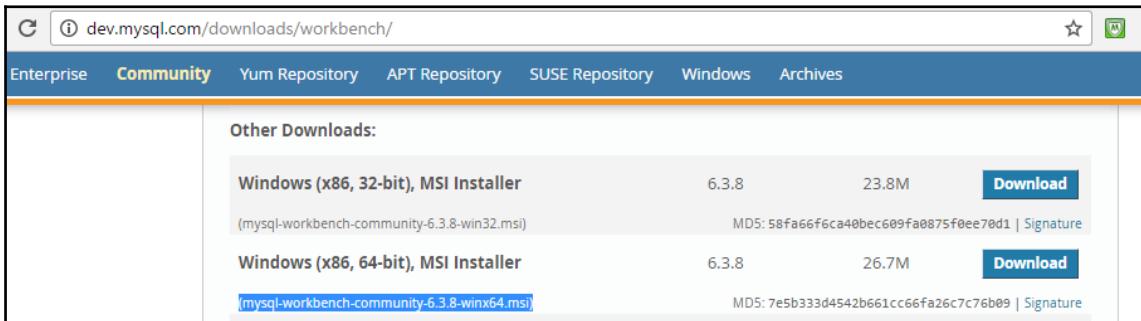
The screenshot shows the MySQL.com homepage. At the top, there's a navigation bar with links for Contact MySQL, Login, and Register. Below the navigation, there's a search bar. The main content area features the MySQL logo and the tagline "The world's most popular open source database". A prominent orange banner in the center says "New! MySQL Workbench 6.3 Enhanced Data Migration". Below this, there's a "Download Now »" button and a screenshot of the MySQL Workbench interface. To the left, there's a sidebar with links for MySQL Editions, MySQL Enterprise Edition (including Datasheet PDF and Technical Specification), MySQL Database, Oracle Enterprise Manager, Enterprise Monitor, Enterprise Backup, Enterprise HA, Enterprise Scalability, and Enterprise Authentication. At the bottom right, there's a link to "MySQL Workbench Home".



Il *MySQL Workbench* è una GUI in sé, molto simile a quella che abbiamo sviluppato nelle ricette precedenti. Viene fornito con alcune funzionalità aggiuntive specifiche per lavorare con MySQL.

Quando hai installato MySQL, se avevi già installato i componenti richiesti sul tuo PC, potresti avere già installato MySQL Workbench. Se non hai installato Workbench, ecco i passaggi per installare MySQL Workbench:

1. Sul [pagina web](http://www.mysql.com/products/workbench/), puoi fare clic sul **Scarica** pulsante, e questo ti porterà al [pagina web](http://dev.mysql.com/downloads/workbench/):



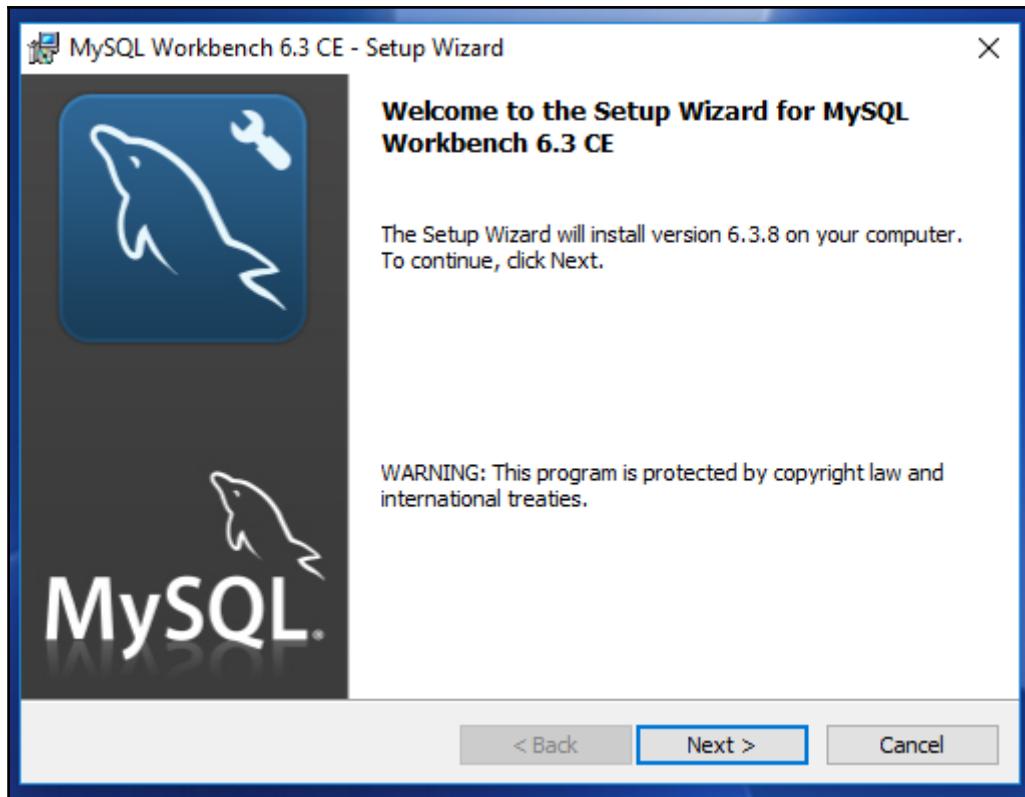
Scarica il programma di installazione MSI che corrisponde al tuo sistema operativo, ad esempio, **mysql-workbench-community-6.3.8-winx64.msi**, per un tipico Windows 10 con sistema operativo a 64 bit.

L'installazione potrebbe chiederti di accedere con il tuo account Oracle, quindi se non ne hai ancora uno, dovrai creare il tuo account sviluppatore Oracle. Non preoccuparti, è gratuito.



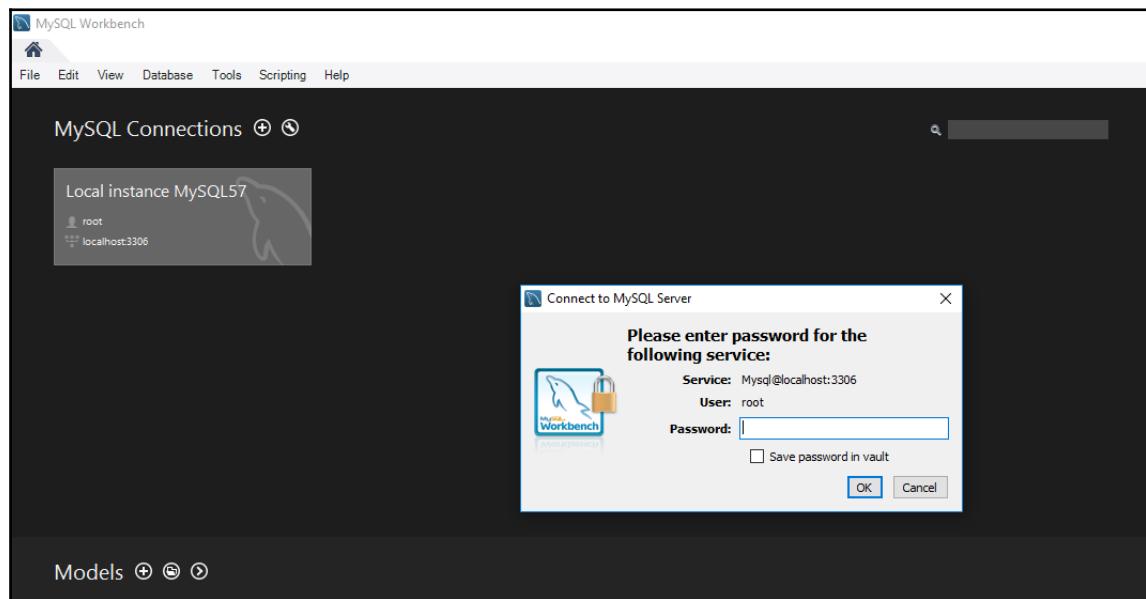
Alcuni anni fa, MySQL è stata acquisita da Oracle. Questo è il motivo per cui è necessario un account Oracle per scaricare e installare MySQL Server e MySQL Workbench.

2. Vedrai la seguente schermata, tra le altre, fino a quando non avrai installato con successo MySQL Workbench:

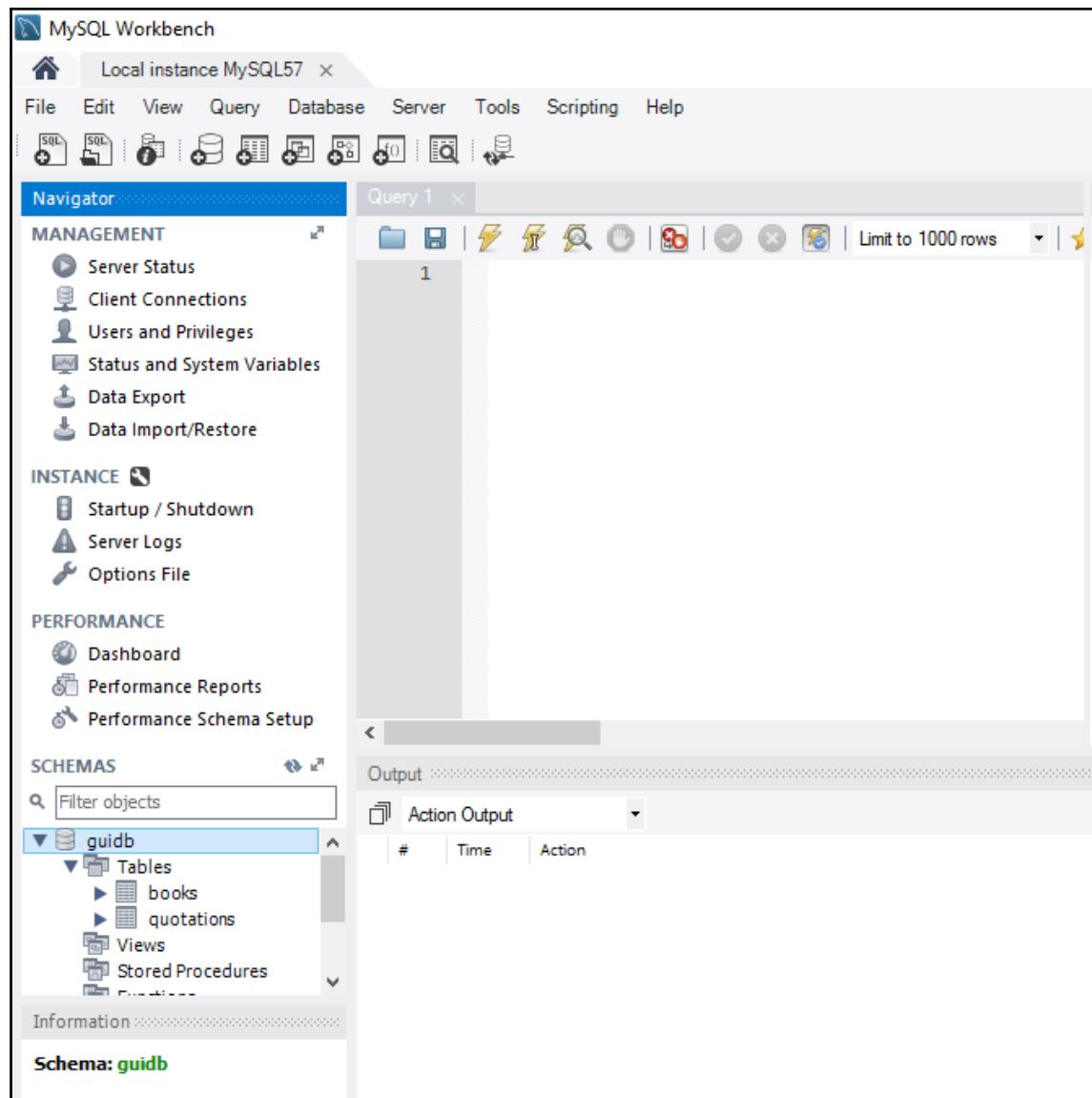


Il 6.3 CE nella finestra di installazione in basso c'è l'abbreviazione di **6.3**
Edizione della Comunità.

Quando avvii MySQL Workbench, ti verrà chiesto di connetterti. Usa l'utente root e la password che hai creato per esso. MySQL Workbench è abbastanza intelligente da riconoscere il tuo server MySQL in esecuzione e la porta su cui è in ascolto:



Una volta effettuato l'accesso con successo alla tua istanza di MySQL Server, possiamo selezionare il nostro guidb:



In basso a sinistra del banco da lavoro, possiamo trovare il nostro **guida** sotto il **SCHEMI** etichetta.

In alcune pubblicazioni e prodotti, i database sono spesso chiamati **SCHEMI**.



Possiamo digitare comandi SQL nell'editor di query ed eseguire i nostri comandi facendo clic sull'icona del fulmine:



I seguenti risultati sono l'editor di query nella griglia dei risultati. Possiamo fare clic sulle diverse schede per vedere i diversi risultati:

A screenshot of the MySQL Workbench application. The interface includes a top menu bar with File, Edit, View, Query, Database, Server, Tools, Scripting, and Help. Below the menu is a toolbar with various icons. On the left is a Navigator pane showing the database schema with "guidb" selected, containing tables like "books" and "quotations", and other objects like Views, Stored Procedures, and Functions. The main area has a "Query 1" tab open, displaying a query editor with the following SQL code:

```
1 USE guidb;
2 SELECT * FROM books;
3 SELECT * FROM quotations;
```

Below the query editor is a "Result Grid" tab showing a table with columns Book_ID, Book_Title, and Book_Page. One row is visible: Book_ID 1, Book_Title "The Meaning of Life", and Book_Page 42. To the right of the result grid are buttons for Filter Rows, Edit, Export/Import, and Wrap Cell Content. At the bottom of the main area is an "Output" pane showing the execution log:

#	Time	Action	Message
1	18:44:45	USE guidb	0 row(s) affected
2	18:44:45	SELECT * FROM books LIMIT 0, 1000	1 row(s) returned
3	18:44:45	SELECT * FROM quotations LIMIT 0, 1000	1 row(s) returned

The bottom navigation bar includes tabs for Management, Schemas, and Information, with "Schemas" currently selected.

Come funziona...

Ora possiamo connetterci al nostro database MySQL tramite il *Banco di lavoro MySQL GUI*. Possiamo eseguire gli stessi comandi SQL che abbiamo emesso prima e ottenere gli stessi risultati che abbiamo quando li abbiamo eseguiti nella nostra GUI Python.

C'è più...

Con la conoscenza che abbiamo acquisito attraverso le ricette all'interno di questo e dei capitoli precedenti, siamo ora ben posizionati per creare le nostre GUI scritte in Python, che possono connettersi e parlare con i database MySQL.

8

Internazionalizzazione e sperimentazione

In questo capitolo, internazionalizzeremo e testeremo la nostra GUI Python, coprendo le seguenti ricette:

- Visualizzazione del testo del widget in diverse lingue
- Modifica dell'intera lingua della GUI contemporaneamente
- Localizzazione della GUI
- Preparare la GUI per l'internazionalizzazione
- Come progettare una GUI in modo agile È
- necessario testare il codice della GUI?
- Impostazione degli orologi di debug
- Configurazione di diversi livelli di output di debug Creazione di codice di
- autotest utilizzando __di Python principale__ sezione Creazione di GUI robuste
- utilizzando i test di unità
- Come scrivere unit test utilizzando l'IDE Eclipse PyDev

introduzione

In questo capitolo, internazionalizzeremo la nostra GUI visualizzando il testo su etichette, pulsanti, schede e altri widget, in diverse lingue. Inizieremo semplicemente e poi esploreremo come possiamo preparare la nostra GUI per l'internazionalizzazione a livello di progettazione.

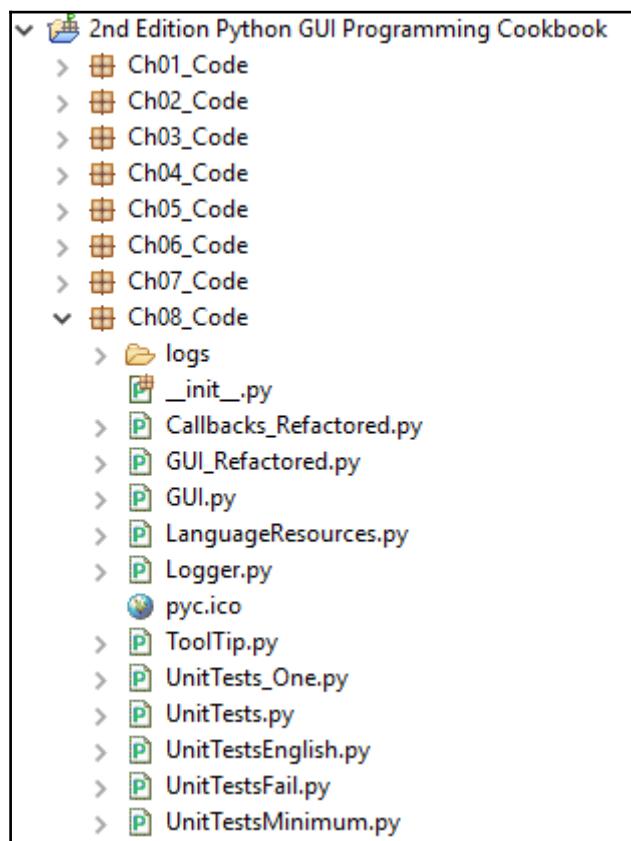
Localizzeremo anche la GUI, che è leggermente diversa dall'internazionalizzazione.



Poiché queste parole sono lunghe, sono state abbreviate per utilizzare il primo carattere della parola, seguito dal numero totale di caratteri tra il primo e l'ultimo carattere, seguito dall'ultimo carattere della parola. Quindi, l'internazionalizzazione diventaI18N e la localizzazione diventaL10N.

Testeremo anche il nostro codice GUI, scriveremo unit test ed esploreremo il valore che i test unitari possono fornire nei nostri sforzi di sviluppo, che ci porteranno alle migliori pratiche di *refactoring* il nostro codice.

Ecco la panoramica dei moduli Python per questo capitolo:



Visualizzazione del testo del widget in diverse lingue

Il modo più semplice per internazionalizzare le stringhe di testo in Python è spostarle in un modulo Python separato e quindi selezionare la lingua da visualizzare nella nostra GUI passando un parametro a questo modulo.

Sebbene questo approccio non sia altamente raccomandato, in base ai risultati della ricerca online, a seconda dei requisiti specifici dell'applicazione che si sta sviluppando, questo approccio potrebbe comunque essere il più pragmatico e veloce da implementare.

Prepararsi

Riutilizzeremo la GUI Python che abbiamo creato in precedenza. Abbiamo commentato una riga di codice Python che crea la scheda MySQL perché non parliamo di un database MySQL in questo capitolo.

Come farlo...

In questa ricetta, inizieremo il I18N della nostra GUI cambiando il titolo di Windows dall'inglese a un'altra lingua.

Poiché il nome GUI è lo stesso in altre lingue, prima espanderemo il nome che ci consentirà di vedere gli effetti visivi delle nostre modifiche.

Quella che segue era la nostra precedente riga di codice:

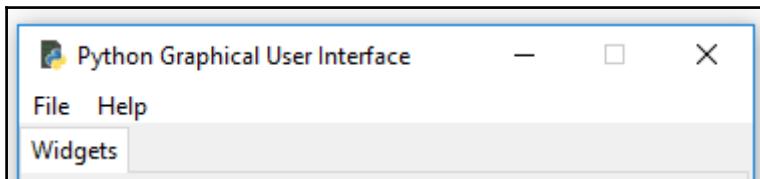
```
self.win.title ("GUI Python")
```

Cambiamo questo nel seguente:

```
self.win.title ("Interfaccia utente grafica Python")
```

La modifica del codice precedente risulta nel seguente titolo per il nostro programma GUI:

GUI_Refactored.py



In questo capitolo, useremo l'inglese e il tedesco per esemplificare il principio dell'internazionalizzazione della nostra GUI Python.

L'hardcoding delle stringhe nel codice non è mai un'idea troppo buona, quindi la prima cosa che possiamo fare per migliorare il nostro codice è separare tutte le stringhe che sono visibili nella nostra GUI in un proprio modulo Python. Questo è l'inizio dell'internazionalizzazione degli aspetti visibili della nostra GUI.



Mentre siamo in I18N, faremo questo refactoring e traduzione linguistica molto positivi, tutto in un unico passaggio.

Creiamo un nuovo modulo Python e chiamiamolo `LanguageResources.py`. Spostiamo quindi la stringa inglese del titolo della nostra GUI in questo modulo e quindi importiamo questo modulo nel codice della nostra GUI.



Stiamo separando la GUI dalle lingue che visualizza, che è un principio di progettazione OOP.

Il nostro nuovo modulo Python, contenente stringhe internazionalizzate, ora ha il seguente aspetto:

```
classe I18N():
    "Internazionalizzazione"
    def __init__(self, language):
        if language == 'en': self.resourceLanguageEnglish() elif language == 'de':
            self.resourceLanguageGerman() else: raise NotImplementedError('Lingua
non supportata.')

    def resourceLanguageEnglish(self):
```

```
    self.title = "Interfaccia utente grafica Python"

def resourceLanguageGerman(self):
    self.title = 'Python Grafische Benutzeroberflaeche'
```

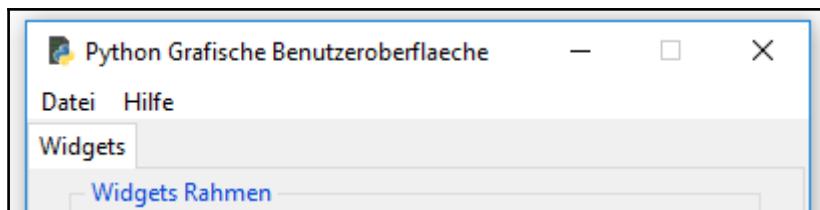
Importiamo questo nuovo modulo Python nel nostro codice GUI Python principale, quindi lo usiamo:

```
da Ch08_Code.LanguageResources import I18N class
OOP():
    def __init__(self):
        self.win = tk.Tk()                                # Crea istanza
        self.i18n = I18N('de')                            # Seleziona la lingua
        self.win.title(self.i18n.title)                   # Aggiungi un titolo
```

A seconda di quale lingua passiamo nel I18N class, la nostra GUI verrà visualizzata in quella lingua.

Eseguendo il codice sopra, ora otteniamo il seguente risultato internazionalizzato:

GUI.py



Come funziona...

Suddividiamo le stringhe hardcoded che fanno parte della nostra GUI in moduli separati. Lo facciamo creando una classe e all'interno del __init__ della classe dentro __init__() metodo, selezioniamo la lingua che verrà visualizzata dalla nostra GUI, a seconda dell'argomento della lingua passato.

Funziona.

Possiamo ulteriormente modularizzare il nostro codice separando le stringhe internazionalizzate in file separati, potenzialmente in XML o in un altro formato. Potremmo anche leggerli da un database MySQL.



Questo è un approccio di codifica della separazione delle preoccupazioni, che è al centro della programmazione OOP.

Cambiare l'intera lingua della GUI, tutto in una volta

In questa ricetta, cambieremo tutti i nomi visualizzati della GUI, tutti in una volta, eseguendo il refactoring di tutte le stringhe inglesi precedentemente codificate in un modulo Python separato e quindi internazionalizzando quelle stringhe.

Questa ricetta mostra che è un buon principio di progettazione evitare di codificare le stringhe visualizzate dalla nostra GUI, ma separare il codice della GUI dal testo visualizzato dalla GUI.



Progettare la nostra GUI in modo modulare rende l'internazionalizzazione molto più semplice.

Prepararsi

Continueremo a utilizzare la GUI della ricetta precedente. In quella ricetta avevamo già internazionalizzato il titolo della GUI.

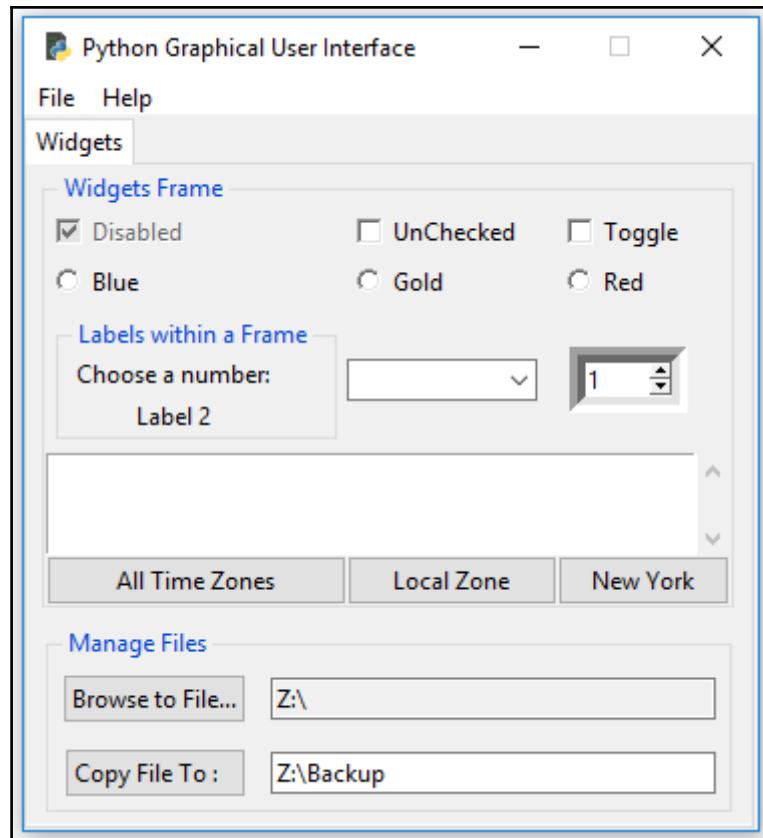
Come farlo...

Per internazionalizzare il testo visualizzato in tutti i nostri widget della GUI, dobbiamo spostare tutte le stringhe codificate in un modulo Python separato, e questo è ciò che faremo dopo.

In precedenza, le stringhe di parole visualizzate dalla nostra GUI erano sparse in tutto il nostro codice Python.

Ecco come appariva la nostra GUI senza I18N:

`GUI_Refactored.py`



Ogni singola stringa di ogni widget, incluso il titolo della nostra GUI, i nomi dei controlli delle schede e così via, erano tutte codificate e mescolate con il codice che crea la GUI.



È una buona idea pensare a come possiamo internazionalizzare al meglio la nostra GUI nella fase di progettazione del nostro processo di sviluppo del software GUI.

Quello che segue è un estratto di come appare il nostro codice:

```
WIDGET_LABEL = 'Widget Frame' class
OOP():
    def __init__(self):
        self.win = tk.Tk()                                # Crea istanza
        self.win.title ("GUI Python")                      # Aggiungi un titolo
```

```
# Funzione di richiamata del pulsante radio
def radCall(self):
    radSel=self.radVar.get() if radSel == 0:
        self.monty2.configure(text='Blue') elif radSel == 1:
        self.monty2.configure(text='Gold') elif radSel == 2 :
        self.monty2.configure(text='Rosso')
```



In questa ricetta, stiamo internazionalizzando tutte le stringhe visualizzate nei nostri widget della GUI. Non stiamo internazionalizzando il testo *entrato* nella nostra GUI, perché questo dipende dalle impostazioni locali del tuo PC.

Quello che segue è il codice per le stringhe internazionalizzate in inglese:

```
classeI18N():
    "Internazionalizzazione"
    def __init__(self, lingua):
        if language == 'en': self.resourceLanguageEnglish() elif language == 'de':
            self.resourceLanguageGerman() else: raiseNotImplementedError('Lingua
non supportata.')

    def resourceLanguageEnglish(self):
        self.title = "Interfaccia utente grafica Python"

        self.file = "File"
        self.new = "Nuovo"
        self.exit = "Esci"
        self.help = "Aiuto"
        self.about = "Informazioni"

        self.WIDGET_LABEL = ' Cornice widget '

        self.disabled = "Disabilitato"
        self.unChecked = "Non selezionato"
        self.toggle = "Commuta"

    # Elenco radiobutton
    self.colors = ["Blu", "Oro", "Rosso"]
    self.colorsIn = ["in blu", "in oro", "in rosso"]

    self.labelsFrame = ' Etichette all'interno di una cornice '
    self.chooseNumber = "Scegli un numero:" self.label2 =
    "Etichetta 2"

    self.mngrFiles = 'Gestisci file'

    self.browseTo = "Sfoglia su file..." self.copyTo =
    "Copia file in: "
```

Nel nostro modulo della GUI Python, tutte le stringhe precedentemente codificate sono ora sostituite da un'istanza del nostro nuovo I18N classe, che risiede nel LanguageResources.py modulo.

Ecco un esempio dal nostro refactoring GUI.py modulo:

```
da Ch08_Code.LanguageResources import I18N class
OOP():
    def __init__(self):
        self.win = tk.Tk()                                     # Crea istanza
        self.i18n = I18N('de')                                # Seleziona la lingua
        self.win.title(self.i18n.title) # Aggiungi un titolo

    # Funzione di richiamata del pulsante radio
    def radCall(self):
        radSel = self.radVar.get() if radSel == 0:
            self.widgetFrame.configure(text=
                self.i18n.WIDGET_LABEL + self.i18n.colorsIn[0])
        elif radSel == 1: self.widgetFrame.configure(text=
            self.i18n.WIDGET_LABEL + self.i18n.colorsIn[1])
        elif radSel == 2: self.widgetFrame.configure(text=
            self.i18n.WIDGET_LABEL + self.i18n.colorsIn[2])
```

Nota come tutte le stringhe inglesi precedentemente hardcoded sono state sostituite da chiamate al istanza del nostro nuovo I18N classe. Un esempio è self.win.title(self.i18n.title).

Ciò che questo ci dà è la capacità di internazionalizzare la nostra GUI. Dobbiamo semplicemente usare gli stessi nomi di variabili e combinarli passando un parametro per selezionare la lingua che vogliamo visualizzare.

Potremmo anche cambiare le lingue al volo come parte della GUI, oppure potremmo leggere le impostazioni del PC locale e decidere quale lingua deve essere visualizzata dal nostro testo della GUI in base a tali impostazioni.



Un esempio di come leggere le impostazioni locali è trattato nella prossima ricetta, *Localizzazione della GUI*.

Ora possiamo implementare la traduzione in tedesco semplicemente compilando i nomi delle variabili con le parole corrispondenti:

```
classe I18N():
    "Internazionalizzazione" def
    __init__(self, language):
        if language == 'en': self.resourceLanguageEnglish() elif language ==
            'de': self.resourceLanguageGerman()
```

```
else: raise NotImplementedError('Lingua non supportata.')

def resourceLanguageGerman(self):
    self.file = "Data"
    self.new = "Nuovo"
    self.exit = "Schliessen" self.help =
    "Hilfe"
    self.about = "Ueber"

    self.WIDGET_LABEL = ' Widget Rahmen '

    self.disabled = "Deaktiviert" self.unChecked
    = "Nicht Markiert" self.toggle = "Markieren"

    # Elenco radiobutton
    self.colors = ["Blau", "Gold", "Rot"]
    self.colorsIn = ["in Blau", "in Gold", "in Rot"]

    self.labelsFrame = ' Etiketten im Rahmen '
    self.chooseNumber = "Waehle eine Nummer:" self.label2
    = "Etikette 2"

    self.mgrFiles = ' Dateien Organisieren '

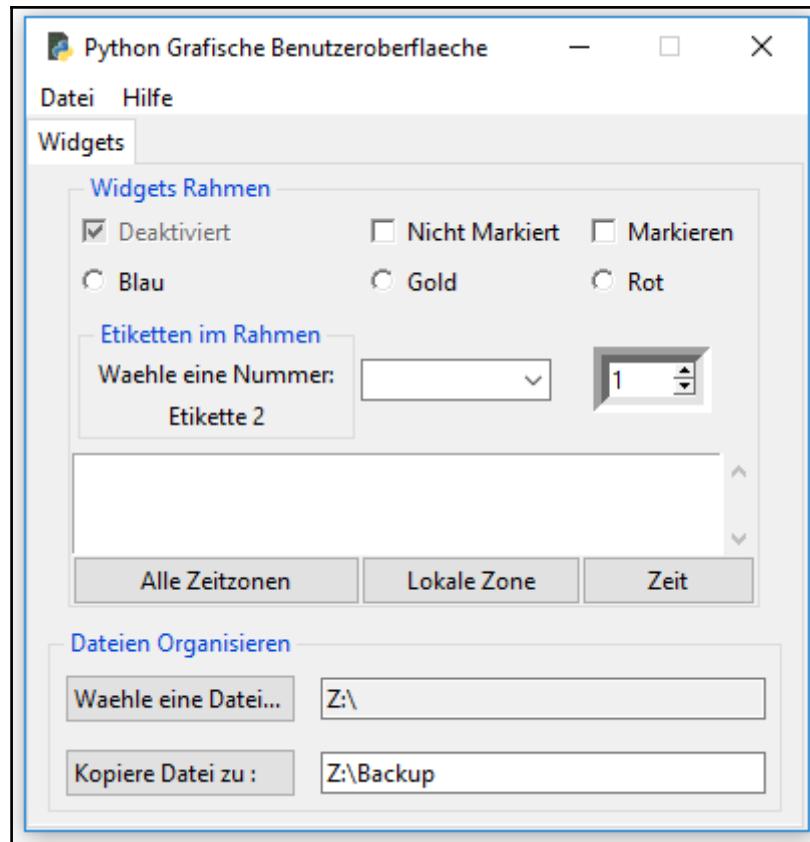
    self.browseTo = "Waehle eine Datei... " self.copyTo =
    "Kopiere Datei zu : "
```

Nel nostro codice GUI, ora possiamo cambiare l'intera lingua di visualizzazione della GUI in una riga di codice Python:

```
classOOP():
    def __init__(self):
        self.win = tk.Tk()                      # Crea istanza
        self.i18n = I18N('de')                  # Passa in lingua
```

L'esecuzione del codice precedente crea la seguente GUI internazionalizzata:

GUI.py



Come funziona...

Per internazionalizzare la nostra GUI, abbiamo rifactorizzato le stringhe hardcoded in un modulo separato e poi abbiamo usato gli stessi membri della classe per internazionalizzare la nostra GUI passando una stringa come inizializzatore del nostro I18N classe, controllando efficacemente la lingua visualizzata dalla nostra GUI.

Localizzare la GUI

Dopo il primo passaggio di internazionalizzazione della nostra GUI, il passaggio successivo è localizzarlo. Perché desideriamo farlo?

Bene, qui negli Stati Uniti d'America siamo tutti cowboy e viviamo in fusi orari diversi.

Quindi, mentre siamo internazionalizzati negli Stati Uniti, i nostri cavalli si svegliano in diversi fusi orari (e si aspettano di essere nutriti secondo il loro programma di fuso orario interno del cavallo).

È qui che entra in gioco la localizzazione.

Prepararsi

Stiamo estendendo la GUI che abbiamo sviluppato nella ricetta precedente localizzandola.

Come farlo...

Iniziamo installando prima Python pytz modulo fuso orario, utilizzando pip. Digitiamo il seguente comando in un prompt del processore dei comandi:

pip install pytz



In questo libro, stiamo usando Python 3.6, che viene fornito con il pip modulo integrato. Se stai utilizzando una versione precedente di Python, potrebbe essere necessario installare il pip modulo prima.

In caso di successo, otteniamo il seguente risultato:

A screenshot of a Windows Command Prompt window titled "Administrator: Command Prompt". The window shows the command "C:\WINDOWS\system32>pip install pytz" being run. The output indicates that pytz is being collected, downloaded (progress bar at 100%), and successfully installed. The final message says "Successfully installed pytz-2016.10".

```
C:\WINDOWS\system32>pip install pytz
Collecting pytz
  Downloading pytz-2016.10-py2.py3-none-any.whl (483kB)
    100% |██████████| 491kB 1.2MB/s
Installing collected packages: pytz
Successfully installed pytz-2016.10

C:\WINDOWS\system32>
```



Lo screenshot precedente mostra che il comando ha scaricato il file .quando formato. Se non lo hai fatto, potresti dover installare Python ruota modulo prima.

Questo ha installato Python pytz modulo in pacchetti-sito cartella, quindi ora possiamo importare questo modulo dal nostro codice della GUI Python.

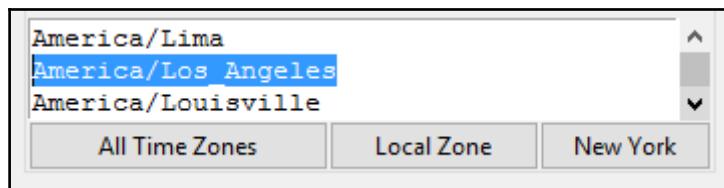
Possiamo elencare tutti i fusi orari esistenti eseguendo il seguente codice, che visualizzerà i fusi orari nel nostro Testo scorrevole aggeggio. Per prima cosa, aggiungiamo un nuovoPulsante widget alla nostra GUI:

```
import pytz
classe OOP():
    # Tasto TZ richiamata
    def allTimeZones(self):
        per tz in pytz.all_timezones:
            self.scr.insert(tk.INSERT, tz + '\n')

    def createWidgets(self):
        # Aggiunta di un pulsante TZ
        self.allTZs = ttk.Button(self.widgetFrame,
                               text=self.i18n.timeZones,
                               comando=self.allTimeZones)
        self.allTZs.grid(colonna=0, riga=9, appiccicoso='NOI')
```

Facendo clic sul nostro nuovo Pulsante widget risulta nel seguente output:

GUI.py



Dopo aver installato il tzlocal Python, possiamo stampare la nostra locale corrente eseguendo il seguente codice:

```
# TZ Local Button callback
def localZone(self):
    da tzlocal import get_localzone self.scr.insert(tk.INSERT,
get_localzone())

def createWidgets(self):
```

```
# Aggiunta del pulsante TZ locale
self.localTZ = ttk.Button(self.widgetFrame,
                           text=self.i18n.localZone,
                           comando=self.localZone
                           self.localTZ.grid(column=1, row=9, sticky='NOI')
```

Abbiamo internazionalizzato le corde dei nostri due nuovi pulsanti nel Risorse.py.

Versione inglese:

```
self.timeZones = "Tutti i fusi orari"
self.localZone = "Zona locale"
```

versione tedesca:

```
self.timeZones = "Tutte le zone"
self.localZone = "Zona locale"
```

Fare clic sul nostro nuovo pulsante ora ci dice in quale fuso orario ci troviamo (ehi, non lo sapevamo, vero...).

GUI.py

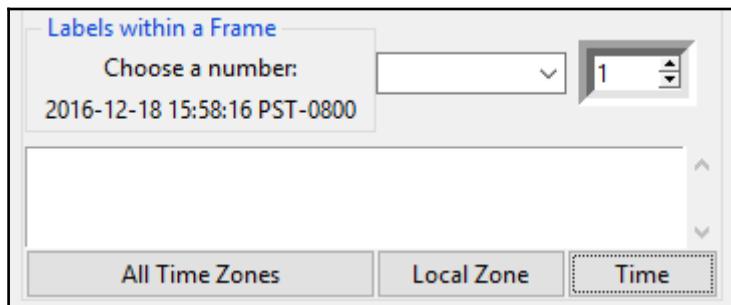


Ora possiamo tradurre la nostra ora locale in un fuso orario diverso. Usiamo l'ora solare degli Stati Uniti come esempio.

Mostriamo la nostra ora locale attuale nella nostra Label 2 inutilizzata migliorando il nostro codice esistente.

Quando eseguiamo il codice, la nostra Label 2 internazionalizzata (visualizzata come Etichetta 2 in tedesco) visualizzerà l'ora locale attuale:

GUI.py



Ora possiamo cambiare la nostra ora locale in US EST convertendola prima in **Tempo universale coordinato (UTC)** e quindi applicando il fuso orario funzione dall'importato pytz modulo:

```
import pytz
classe OOP():

    # Formatta l'ora locale degli Stati Uniti con le informazioni sul fuso orario
    def getDate(self):
        fmtStrZone = "%Y-%m-%d %H:%M:%S %Z%z"
        # Ottieni tempo universale coordinato
        utc = datetime.now(timezone('UTC'))
        print(utc.strftime(fmtStrZone))

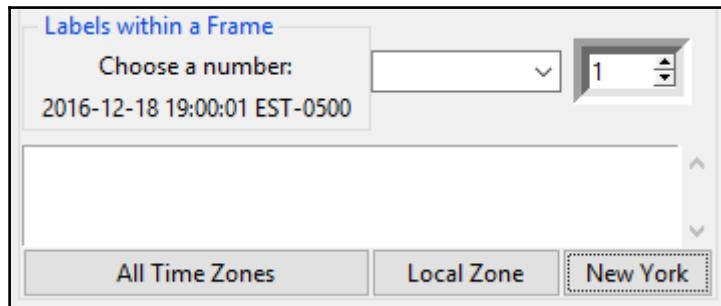
        # Converti oggetto datetime UTC in Los Angeles TimeZone
        la = utc.astimezone(timezone('America/Los_Angeles'))
        print(la.strftime(fmtStrZone))

        # Converti oggetto datetime UTC in New York TimeZone
        ny = utc.astimezone(timezone('America/New_York'))
        print(ny.strftime(fmtStrZone))

        # aggiorna l'etichetta della GUI con l'ora e il fuso di New York
        self.lbl2.set(ny.strftime(fmtStrZone))
```

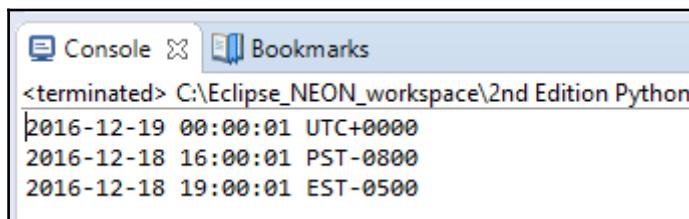
Facendo clic sul pulsante, ora rinominato New York, si ottiene il seguente output:

GUI.py



La nostra etichetta 2 è stata aggiornata con l'ora corrente di New York e stiamo stampando gli orari UTC delle città, Los Angeles e New York, con le rispettive conversioni di fuso orario, relativi all'ora UTC sulla console Eclipse, utilizzando una data statunitense stringa di formattazione:

GUI.py



UTC non osserva mai l'ora legale. Durante **Ora legale orientale (EDT)** UTC è quattro ore avanti e durante **Ora solare (EST)** è cinque ore avanti rispetto all'ora locale.

Come funziona...

Per localizzare le informazioni su data e ora, dobbiamo prima convertire la nostra ora locale in ora UTC. Quindi applichiamo il fuso orario informazioni e utilizzare il astimezone funzione da pytz Modulo fuso orario Python per convertire in qualsiasi fuso orario in tutto il mondo!

In questa ricetta, abbiamo convertito l'ora locale della costa occidentale degli Stati Uniti in UTC e quindi visualizzato l'ora della costa orientale degli Stati Uniti nell'etichetta 2 della nostra GUI.

Preparazione della GUI per l'internazionalizzazione

In questa ricetta prepareremo la nostra GUI per l'internazionalizzazione rendendoci conto che non tutto è così facile come ci si potrebbe aspettare quando si traduce l'inglese in lingue straniere.

Abbiamo ancora un problema da risolvere, ovvero come visualizzare correttamente i caratteri Unicode non inglesi dalle lingue straniere.

Ci si potrebbe aspettare che la visualizzazione dei caratteri tedeschi ä, ö e ü Unicode umlaut venga gestita automaticamente da Python 3.6, ma non è così.

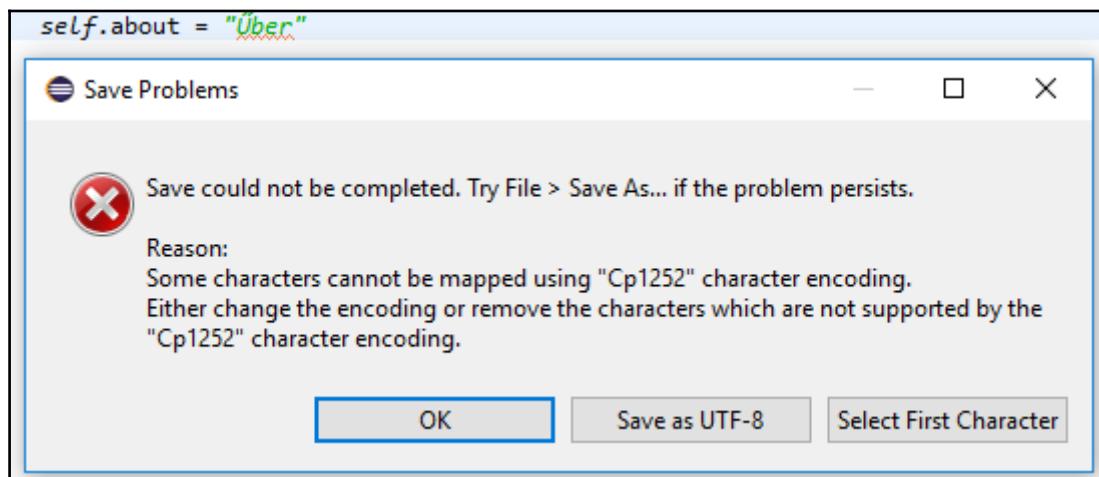
Prepararsi

Continueremo a utilizzare la GUI Python che abbiamo sviluppato nei capitoli recenti. Innanzitutto, cambieremo la lingua predefinita in tedesco nelGUI.py codice di inizializzazione.

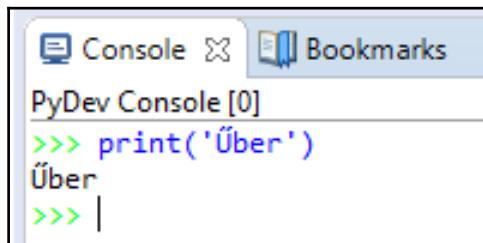
Lo facciamo decommentando la riga, self.i18n = I18N('de').

Come farlo...

Quando cambiamo la parola Ueber al tedesco corretto ber usando il carattere dieresi il plugin Eclipse PyDev non è troppo felice:

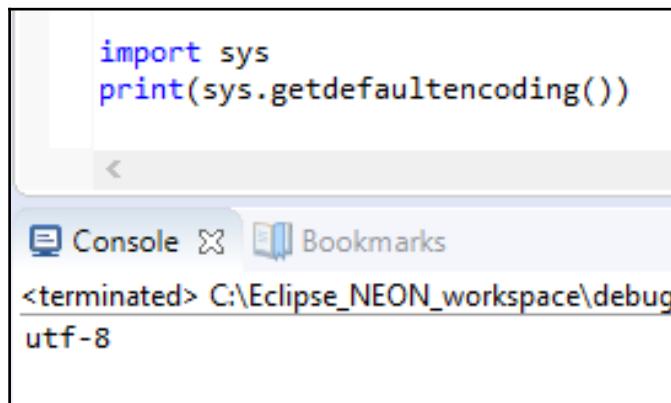


Riceviamo un messaggio di errore, che è un po' confuso perché, quando eseguiamo la stessa riga di codice dalla console di Eclipse PyDev, otteniamo il risultato previsto:



```
Console Bookmarks  
PyDev Console [0]  
>>> print('Über')  
Über  
>>> |
```

Quando chiediamo la codifica predefinita di Python otteniamo il risultato atteso, che è utf-8:

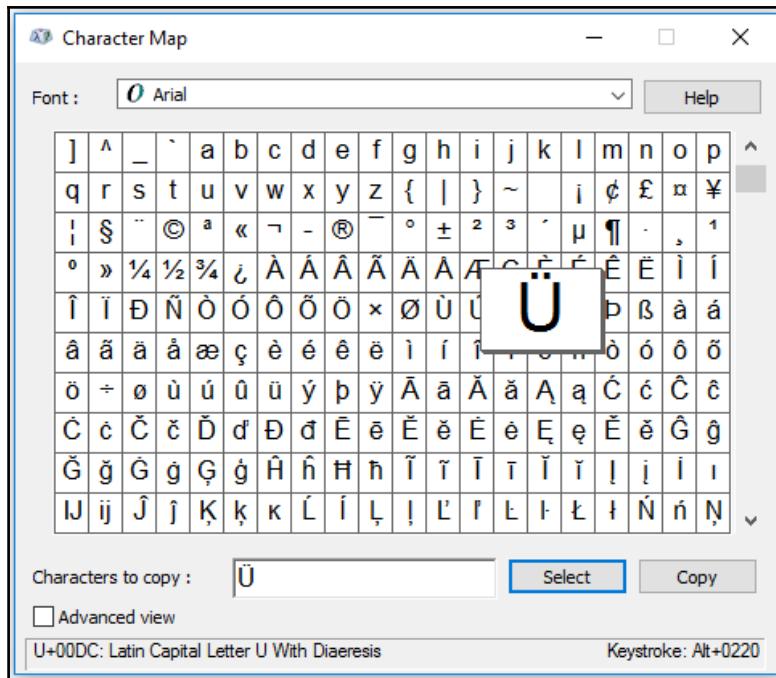


```
import sys  
print(sys.getdefaultencoding())  
  
<  
Console Bookmarks  
<terminated> C:\Eclipse_NEON_workspace\debug  
utf-8
```

Possiamo, ovviamente, ricorrere sempre alla rappresentazione diretta di Unicode.



Usando la mappa dei caratteri integrata di Windows, possiamo trovare la rappresentazione Unicode del carattere dieresi, che è **U+00DC** per la capitale **tu** con dieresi:



Sebbene questa soluzione alternativa sia davvero brutta, fa il trucco. Invece di digitare il carattere letterale Ü, possiamo passare in Unicode di **U+00DC** per visualizzare correttamente questo carattere nella nostra GUI:

```
print("\u00DC" + "ber")
```

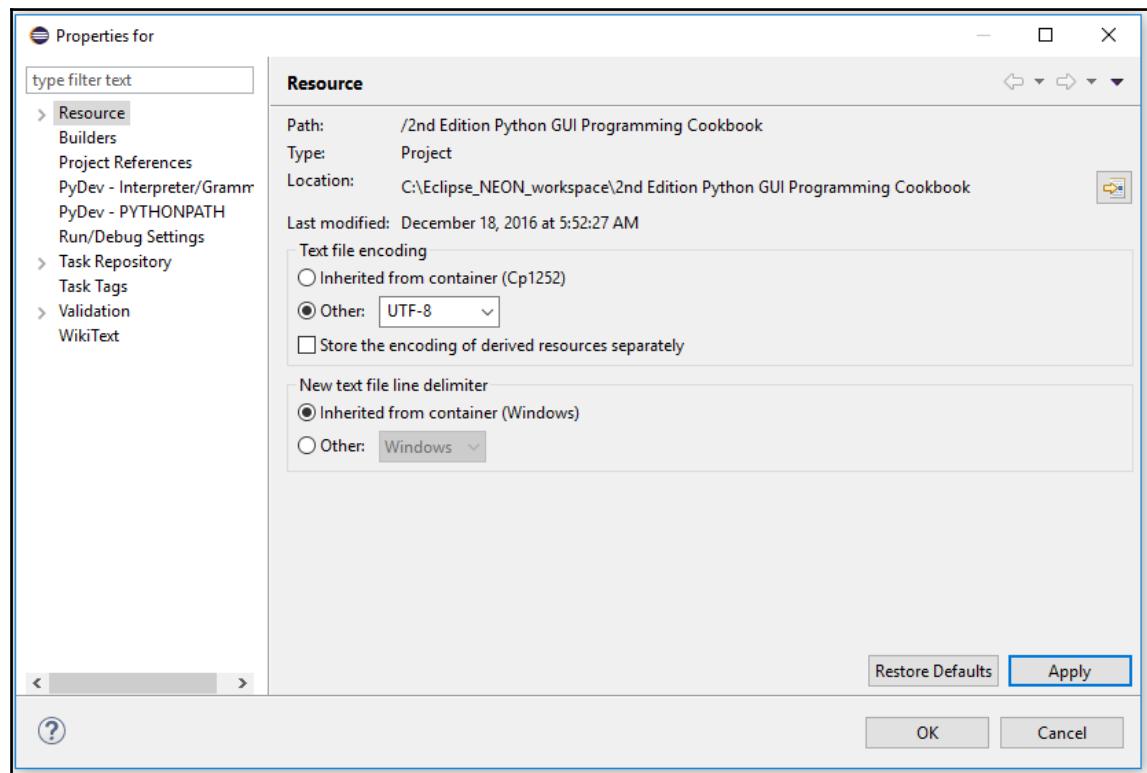
The screenshot shows an Eclipse PyDev console window. The code `print("\u00DC" + "ber")` is entered in the top text area. The output in the console shows the result: `<terminated> C:\Eclipse_NEON_workspace Über`. The word "Über" is correctly displayed with the umlaut character.

Possiamo anche accettare la modifica nella codifica predefinita da *Cp1252* per **UTF-8** usando PyDev con Eclipse ma potremmo non sempre ottenere il prompt per farlo.

Invece, potremmo vedere il seguente messaggio di errore visualizzato:

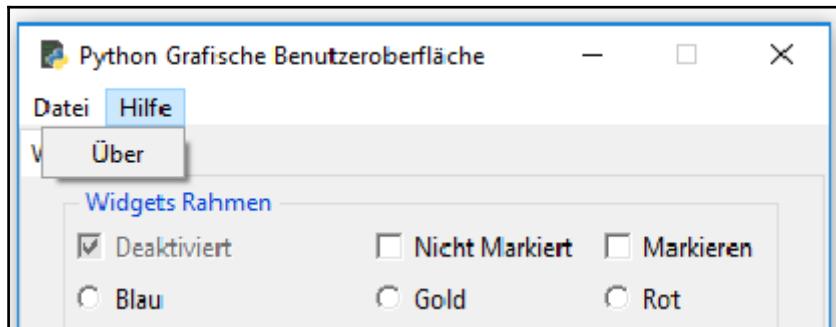
A screenshot of the Eclipse IDE's Console view. The console window title is "Console". It shows the output of a Python script named "default_encoding.py" which contains the line `print('Über')`. The output text is displayed in green and orange colors. Below the code, the message "`<terminated> C:\Eclipse_NEON_workspace\debug\default_encoding.py`" is shown. Underneath that, a red error message reads: `File "C:\Eclipse_NEON workspace\debug\default encoding"` followed by `SyntaxError: Non-UTF-8 code starting with '\xdc' in file`.

Il modo per risolvere questo problema è cambiare il progetto PyDev **Codifica file di testo** proprietà a **UTF-8**:



Dopo aver modificato la codifica predefinita di PyDev, ora possiamo visualizzare quei caratteri dieresi tedeschi. Abbiamo anche aggiornato il titolo per utilizzare il carattere ä tedesco corretto:

GUI_Refactored.py



Come funziona...

L'internazionalizzazione e il lavoro con i caratteri Unicode di lingue straniere spesso non sono così semplici come vorremmo. A volte dobbiamo trovare soluzioni alternative ed esprimere caratteri Unicode tramite Python usando la rappresentazione diretta anteponendo \u può fare il trucco.

Altre volte, dobbiamo solo trovare le impostazioni del nostro ambiente di sviluppo da regolare.

Come progettare una GUI in modo agile

Il moderno approccio di sviluppo software agile alla progettazione e alla codifica è scaturito dalle lezioni apprese dai professionisti del software. Questo metodo si applica a una GUI tanto quanto a qualsiasi altro codice. Una delle chiavi principali dello sviluppo software agile è il processo di refactoring applicato continuamente.

Un esempio pratico di come il refactoring del nostro codice può aiutarci nel nostro lavoro di sviluppo software consiste nell'implementare prima alcune semplici funzionalità utilizzando le funzioni.

Man mano che il nostro codice cresce in complessità, potremmo voler rifattorizzare le nostre funzioni in metodi di una classe. Questo approccio ci consentirebbe di rimuovere le variabili globali e anche di essere più flessibili su dove posizionare i metodi all'interno della classe.

Mentre la funzionalità del nostro codice non è cambiata, la struttura sì.

In questo processo, codice, test, refactoring e quindi test di nuovo. Lo facciamo in cicli brevi e spesso iniziamo con il codice minimo richiesto per far funzionare alcune funzionalità.



Lo sviluppo di software basato sui test è uno stile particolare della metodologia di sviluppo agile.

Mentre la nostra GUI funziona bene, il nostro principale GUI.py il codice è diventato sempre più complesso e ha iniziato a diventare un po' più difficile mantenere una panoramica del nostro codice. Ciò significa che dobbiamo rifattorizzare il nostro codice.

Prepararsi

Rifaremo la GUI che abbiamo creato nei capitoli precedenti. Useremo la versione inglese della GUI.

Come farlo...

Abbiamo già suddiviso tutti i nomi visualizzati dalla nostra GUI quando l'abbiamo internazionalizzata nella ricetta precedente. È stato un ottimo inizio per il refactoring del nostro codice.



Il refactoring è il processo di miglioramento della struttura, della leggibilità e della manutenibilità del codice esistente. Non stiamo aggiungendo nuove funzionalità.

Nei capitoli e nelle ricette precedenti, abbiamo esteso la nostra GUI in un approccio di sviluppo a cascata dall'alto verso il basso, aggiungendo importare verso l'alto e il codice verso il basso del codice esistente.

Sebbene ciò fosse utile quando si esaminava il codice, ora sembra un po' disordinato e possiamo migliorarlo per aiutare il nostro sviluppo futuro.

Prima ripuliamo il nostro importare sezione delle dichiarazioni, che attualmente ha il seguente aspetto:

```
# =====
# importazioni
# =====
importa tkinter come tk da
tkinter importa ttk
da tkinter import scrolledtext
```

```
from tkinter import Menu from tkinter
import Spinbox import
Ch08_Code.ToolTip as tt from threading
import Thread da time import sleep

dalla coda importazione coda Que
from tkinter import filedialog as fd from os
import path
da tkinter import messagebox come mBox
da Ch08_Code.LanguageResources importa I18N da
datetime importa datetime
da pytz import all_timezones, timezone

# Livello del modulo GLOBALE
GLOBAL_CONST = 42
```

Raggruppando semplicemente le importazioni correlate, possiamo ridurre il numero di righe di codice, il che migliora la leggibilità delle nostre importazioni, facendole apparire meno opprimenti:

```
# =====
# importazioni
# =====
importa tkinter come tk
from tkinter import ttk, scrolledtext, Menu, Spinbox, filedialog as fd,
                    messagebox come mBox

dalla coda di importazione Coda dal percorso di
importazione del sistema operativo
import Ch08_Code.ToolTip come tt
da Ch08_Code.LanguageResources import I18N da
Ch08_Code.Logger import Logger, LogLevel

# Livello del modulo GLOBALE
GLOBAL_CONST = 42
```

Possiamo rifattorizzare ulteriormente il nostro codice suddividendo i metodi di callback nei propri moduli. Ciò migliora la leggibilità separando le diverse istruzioni di importazione nei moduli in cui sono richieste.

Rinominiamo il nostro GUI.py come GUI_Refactored.py e creare un nuovo modulo, che noi nome Callbacks_Refactored.py.

Questo ci dà questa nuova architettura:

```
# =====
# importazioni
# =====
importa tkinter come tk
from tkinter import ttk, scrolledtext, Menu, Spinbox,
```

```
filedialog come fd, messagebox come mBox
dalla coda di importazione Coda dal percorso di
importazione del sistema operativo
import Ch08_Code.ToolTip come tt
da Ch08_Code.LanguageResources import I18N da
Ch08_Code.Logger import Logger, LogLevel
da Ch08_Code.Callbacks_Refactored import Callbacks

# Livello del modulo GLOBALI
GLOBAL_CONST = 42

classe OOP():
    def __init__(self):
        # Metodi di callback ora in un modulo diverso
        self.callBacks = Richiamate(self)
```

Nota come stiamo passando un'istanza della nostra classe GUI (se stesso) quando chiami il Richiamate inizializzatore.

Il nostro nuovo Richiamate classe è la seguente:

```
# =====
# importazioni
# =====
import tkinter as tk from time
import sleep
da threading import Thread
from pytz import all_timezones, timezone from
datetime import datetime

Richiamate di classe():
    def __init__(self, ops):
        self.oop = oop

    def defaultFileEntries(self):
        self.oop.fileEntry.delete(0, tk.END)
        self.oop.fileEntry.insert(0, 'Z:')                                # percorso fasullo
        self.oop.fileEntry.config(state='readonly')
        self.oop.netwEntry.delete (0, tk.END)
        self.oop.netwEntry.insert(0, 'Z:Backup')                          # percorso fasullo

    # Richiamata casella combinata
    def _combo(self, val=0):
        valore = self.oop.combo.get() self.oop.scr.insert(tk.INSERT,
        valore + '\n')
```

Nell'inizializzatore della nostra nuova classe, l'istanza della GUI passata viene salvata con il nome self.oop e utilizzato in questo nuovo modulo di classe Python.

L'esecuzione del codice della GUI refactoring funziona ancora. Abbiamo solo aumentato la sua leggibilità e ridotto la complessità del nostro codice in preparazione per ulteriori lavori di sviluppo.

Come funziona...

Abbiamo prima migliorato la leggibilità del nostro codice raggruppando le relative istruzioni di importazione. Successivamente abbiamo suddiviso i metodi di callback nella loro classe e modulo, al fine di ridurre ulteriormente la complessità del nostro codice.

Avevamo già adottato lo stesso approccio OOP avendo il Descrizione comando risiede nel proprio modulo e internazionalizzando tutte le stringhe della GUI nelle ricette precedenti. In questa ricetta, abbiamo fatto un ulteriore passo avanti nel refactoring passando la nostra istanza nella classe del metodo di callback su cui si basa la nostra GUI. Questo ci consente di utilizzare tutti i nostri widget della GUI.



Ora che abbiamo compreso meglio il valore di un approccio modulare allo sviluppo del software, molto probabilmente inizieremo con questo approccio nei nostri futuri progetti di software.

Dobbiamo testare il codice della GUI?

Testare il nostro software è un'attività importante durante la fase di codifica, nonché durante il rilascio di service pack o correzioni di bug.

Ci sono diversi livelli di test. Il primo livello è il test dello sviluppatore, che spesso inizia con il compilatore o l'interprete che non ci consente di eseguire il nostro codice difettoso, costringendoci a testare piccole parti del nostro codice a livello dei singoli metodi.

Questo è il primo livello di difesa.

Un secondo livello di codifica in modo difensivo è quando il nostro sistema di controllo del codice sorgente ci informa di alcuni conflitti da risolvere e non ci consente di controllare il nostro codice modificato.

Questo è molto utile e assolutamente necessario quando lavoriamo professionalmente in un team di sviluppatori. Il sistema di controllo del codice sorgente è nostro amico e indica le modifiche che sono state impegnate in un particolare ramo o cima dell'albero, da noi stessi o da altri nostri sviluppatori, e ci dice che la nostra versione locale del codice è sia obsoleta che ha alcuni conflitti che devono essere risolti prima di poter inviare il nostro codice al repository.

Questa parte presuppone che tu usi un sistema di controllo del codice sorgente per gestire e archiviare il tuo codice. Gli esempi includono git, mercurial, svn e molti altri. Git è un controllo del codice sorgente molto popolare ed è gratuito per un singolo utente.

Un terzo livello è il livello delle API in cui incapsuliamo potenziali modifiche future al nostro codice consentendo solo interazioni con il nostro codice tramite interfacce pubblicate.

Un altro livello di test è il test di integrazione, quando metà del ponte che abbiamo finalmente costruito incontra l'altra metà che gli altri team di sviluppo hanno creato e i due non si incontrano alla stessa altezza (diciamo, una metà è finita due metri o iarde più in alto di l'altra metà...).

Poi, c'è il test dell'utente finale. Anche se abbiamo costruito ciò che hanno specificato, non è proprio quello che volevano.

Tutti gli esempi precedenti sono validi motivi per cui dobbiamo testare il nostro codice sia nelle fasi di progettazione che di implementazione.

Prepararsi

Testeremo la GUI che abbiamo creato nelle ricette e nei capitoli recenti. Mostreremo anche alcuni semplici esempi di cosa può andare storto e perché dobbiamo continuare a testare il nostro codice e il codice che chiamiamo tramite API.

Come farlo...

Mentre molti sviluppatori esperti sono cresciuti spruzzando printf() istruzioni su tutto il codice durante il debug, molti sviluppatori nel 21° secolo sono abituati ai moderni ambienti di sviluppo IDE che accelerano in modo efficiente i tempi di sviluppo.

In questo libro, stiamo usando il plugin PyDev Python per l'IDE Eclipse.

Se stai appena iniziando a utilizzare un IDE, come Eclipse con PyDev, all'inizio potrebbe essere un po' travolgente. Lo strumento Python IDLE fornito con Python 3.6 ha un debugger più semplice e potresti volerlo esplorare prima.

Ogni volta che qualcosa va storto nel nostro codice, dobbiamo eseguirne il debug. Il primo passo per fare ciò è impostare i punti di interruzione e quindi scorrere il nostro codice, riga per riga o metodo per metodo.

Entrare e uscire dal nostro codice è un'attività quotidiana finché il codice non funziona senza intoppi.

Nella programmazione della GUI Python, una delle prime cose che può andare storto è perdere l'importazione dei moduli richiesti o l'importazione dei moduli esistenti.

Qui c'è un semplice esempio:

GUI.py con l'istruzione import # importa tkinter come tk commentata:

The screenshot shows the Eclipse IDE interface. In the top editor pane, there is a Python class definition:

```
#=====
class OOP():
    def __init__(self, language='en'):
        # Create instance
        self.win = tk.Tk()
```

The word "tk" is highlighted in red, indicating a syntax error. Below the editor is a terminal window titled "Console". It shows the command line and a traceback:

```
<terminated> C:\Eclipse_NEON_workspace\2nd Edition Python
Traceback (most recent call last):
  File "C:\Eclipse_NEON_workspace\2nd Edition Py
    oop = OOP()
  File "C:\Eclipse_NEON_workspace\2nd Edition Py
    self.win = tk.Tk()
NameError: name 'tk' is not defined
```

Stiamo cercando di creare un'istanza di tkinter classe, ma le cose non funzionano come previsto.

Bene, abbiamo semplicemente dimenticato di importare il modulo e alias esso come tk, e possiamo risolvere questo problema aggiungendo una riga di codice Python sopra la creazione della nostra classe, dove risiedono le istruzioni di importazione:

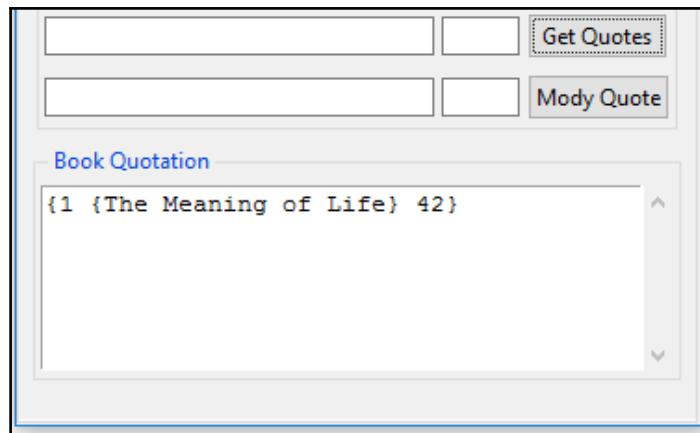
```
# =====
# importazioni
# =====
import tkinter as tk
```

Questo è un esempio in cui il nostro ambiente di sviluppo esegue i test per noi. Non ci resta che eseguire il debug e la correzione del codice.

Un altro esempio più strettamente correlato ai test degli sviluppatori è quando codifichiamo condizionali e, durante il nostro sviluppo regolare, non esercitiamo tutti i rami della logica.

Usando un esempio da capitolo 7, *Memorizzazione dei dati nel nostro database MySQL tramite la nostra GUI*, diciamo che clicchiamo sul **Ottenere le quotazioni** e questo funziona, ma non abbiamo mai cliccato sul **Citazione di moda** pulsante. Il primo clic sul pulsante crea il risultato desiderato ma il secondo genera un'eccezione (perché non avevamo ancora implementato questo codice e probabilmente lo avevamo dimenticato):

GUI_MySQL.py nel capitolo 7, *Memorizzazione dei dati nel nostro database MySQL tramite la nostra GUI*.



Facendo clic su **Citazione di moda** pulsante crea il seguente risultato:

```
Console Bookmarks
C:\Eclipse_NEON_workspace\2nd Edition Python GUI Programming Cookbook\Ch07_Code\GUI_MySQL.py
File "C:\Eclipse_NEON_workspace\2nd Edition Python GUI Programming Cookbook\Ch07_Code\GUI MySQL.py"
raise NotImplementedError("This still needs to be implemented for the SQL command.")
NotImplementedError: This still needs to be implemented for the SQL command.
```

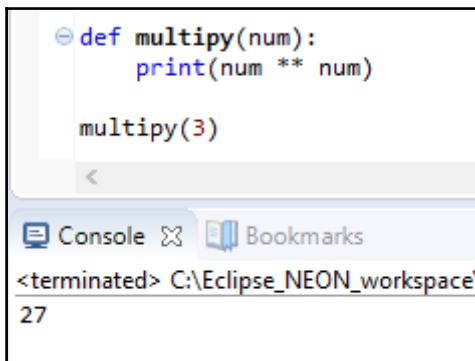
Un'altra potenziale area di bug è quando una funzione o un metodo improvvisamente non restituisce più il risultato previsto. Diciamo che stiamo chiamando la seguente funzione, che restituisce il risultato atteso:

```
def multipy(num):
    print(num * num)

multipy(3)
```

```
Console Bookmarks
<terminated> C:\Eclipse_NEON_workspace\9
```

Quindi, qualcuno commette un errore e non otteniamo più i risultati precedenti:



The screenshot shows a Python script in the Eclipse IDE's code editor. The code defines a function `multipy` that prints the square of its argument. When the function is called with the value 3, instead of printing 9, it prints 27. This is a regression error. Below the code editor is the Eclipse Console view, which shows the command `<terminated> C:\Eclipse_NEON_workspace\` and the number 27.

Invece di moltiplicare, stiamo aumentando per la potenza del numero passato e il risultato non è più quello di una volta.

Nei test del software, questo tipo di bug è chiamato regressione.



Come funziona...

In questa ricetta abbiamo sottolineato l'importanza del test del software durante diverse fasi del ciclo di vita dello sviluppo del software, mostrando diversi esempi di dove il codice può andare storto e introdurre difetti del software (ovvero bug).

Impostazione degli orologi di debug

In moderno **Ambienti di sviluppo integrati (IDE)**, come il plugin PyDev in Eclipse o un altro IDE come NetBeans, possiamo impostare gli orologi di debug per monitorare lo stato della nostra GUI durante l'esecuzione del nostro codice.

Questo è molto simile agli IDE Microsoft di Visual Studio e alle versioni più recenti di Visual Studio.NET.



L'impostazione degli orologi di debug è un modo molto conveniente per aiutare i nostri sforzi di sviluppo.

Prepararsi

In questa ricetta, riutilizzeremo la GUI Python che abbiamo sviluppato nelle ricette precedenti. Noi passerà attraverso il codice che avevamo precedentemente sviluppato e imposteremo gli orologi di debug.

Come farlo...



Sebbene questa ricetta si applichi al plug-in PyDev nell'IDE Eclipse basato su Java, i suoi principi si applicano anche a molti IDE moderni.

La prima posizione in cui potremmo desiderare di posizionare un punto di interruzione è nel punto in cui rendiamo visibile la nostra GUI chiamando il ciclo dell'evento principale di tkinter.

Il simbolo del palloncino verde a sinistra è un punto di interruzione in PyDev/Eclipse. Quando eseguiamo il nostro codice in modalità debug, l'esecuzione del codice verrà interrotta una volta che l'esecuzione raggiunge il punto di interruzione. A questo punto, possiamo vedere i valori di tutte le variabili che sono attualmente nell'ambito. Possiamo anche digitare le espressioni in una delle finestre del debugger, che le eseguirà, mostrandoci i risultati. Se il risultato è quello che vogliamo, potremmo decidere di cambiare il nostro codice usando ciò che abbiamo appena appreso.

Normalmente passiamo attraverso il codice facendo clic su un'icona nella barra degli strumenti del nostro IDE o utilizzando una scorciatoia da tastiera (come premendo *F5* entrare nel codice, *F6* scavalcare, e *F7* per uscire dal metodo corrente):

GUI.py

A screenshot of a Python code editor showing a portion of the file 'GUI.py'. The code is as follows:

```
#=====
# Start GUI
#=====
oop = OOP()
oop.win.mainloop()
```

A green breakpoint marker is placed before the first line of code, indicating where the debugger will stop execution.

Posizionare il punto di interruzione dove l'abbiamo fatto e poi entrare in questo codice risulta essere un problema perché finiamo in qualche tkinter codice che non desideriamo davvero eseguire il debug in questo momento. Usciamo dal basso livello tkinter codice facendo clic sull'icona Step-Out della barra degli strumenti (che è la terza freccia gialla a destra sotto il menu del progetto) o premendo *F7* (supponendo che stiamo usando PyDev in Eclipse).

Abbiamo avviato la sessione di debug facendo clic sull'icona della barra degli strumenti del bug a destra dello screenshot. Se eseguiamo senza eseguire il debug, facciamo clic sul cerchio verde con il triangolo bianco al suo interno, che è l'icona a destra dell'icona del bug:

```

Eclipse_NEON_workspace - Debug - C:\Python36\Lib\tkinter\__init__.py - Eclipse
File Edit Source Refactoring Navigate Search Project Pydev Run Window Help
Debug 2nd Edition Python GUI Programming Cookbook GUI_Refactored.py [Python Run]
  GUI_Refactored.py
    MainThread - pid_12032_id_2267039874240
      mainloop __init__.py:1277
      <module> [GUI_Refactored.py:327]
      ...
LanguageResources Callbacks_Refactored GUI_Refactored Logger
def _bind(self, className, sequence, func, add, 0):
    return self._bind((bind, className), sequence, func, add, 0)
def unbind_class(self, className, sequence):
    """Unbind for all widgets with bindtag CLASSNAME for event SEQUENCE
    all functions."""
    self.tk.call('bind', className, sequence, '')
def mainloop(self, n=0):
    """Call the mainLoop of Tk."""
    self.tk.mainloop(n)
def quit(self):
    """Quit the Tk interpreter. All widgets will be destroyed."""
    self.tk.quit()
def _getints(self, string):
    """Internal function."""
    if string:
        return tuple(map(self.tk.getint, self.tk.splitlist(string)))

```

Un'idea migliore è posizionare il nostro punto di interruzione più vicino al nostro codice per osservare i valori di alcune delle nostre variabili Python. Nel mondo guidato dagli eventi delle moderne GUI, dobbiamo posizionare i nostri punti di interruzione nel codice che viene richiamato durante gli eventi, ad esempio i clic sui pulsanti.

Attualmente, una delle nostre funzionalità principali risiede in un evento di clic del pulsante. Quando facciamo clic sul pulsante etichettato **New York**, creiamo un evento, che, quindi, si traduce in qualcosa che accade nella nostra GUI.

Mettiamo un breakpoint al **New York** metodo di callback del pulsante, che abbiamo chiamato `getTime()`.

Quando ora eseguiamo una sessione di debug, ci fermiamo al punto di interruzione e quindi possiamo abilitare i controlli delle variabili che sono nell'ambito.

Usando PyDev in Eclipse, possiamo fare clic con il pulsante destro del mouse su una variabile e quindi selezionare il comando `watch` dal menu a comparsa. Il nome della variabile, il suo tipo e il valore corrente verranno visualizzati nella finestra di debug delle espressioni mostrata nella schermata successiva. Possiamo anche digitare direttamente nella finestra delle espressioni.

Le variabili che stiamo osservando non si limitano a semplici tipi di dati. Possiamo guardare istanze di classi, elenchi, dizionari e così via.

Quando osserviamo questi oggetti più complessi, possiamo espanderli nella finestra `Espressioni` e approfondire tutti i valori delle istanze di classe, dei dizionari e così via.

Lo facciamo facendo clic sul triangolo a sinistra della nostra variabile osservata che appare più a sinistra sotto il **Nome** colonna, accanto a ciascuna variabile:

`Callbacks_Refactored.py`

Name	Value
> <code>utc</code>	<code>datetime: 2016-12-19 02:18:34.697874+00:00</code>
> <code>la</code>	<code>datetime: 2016-12-18 18:18:34.697874-08:00</code>
> <code>ny</code>	<code>datetime: 2016-12-18 21:18:34.697874-05:00</code>

```

self.oop.scr.delete('1.0', tk.END)
self.oop.scr.insert(tk.INSERT, get_localzone())

# Format local US time with TimeZone info
def getDateTime(self):
    fmtStrZone = "%Y-%m-%d %H:%M:%S %Z%z"
    # Get Coordinated Universal Time
    utc = datetime.now(timezone('UTC'))
    self.oop.log.writeToLog(utc.strftime(fmtStrZone),
                           self.oop.level.MINIMUM)

    # Convert UTC datetime object to Los Angeles TimeZone
    la = utc.astimezone(timezone('America/Los_Angeles'))
    self.oop.log.writeToLog(la.strftime(fmtStrZone),
                           self.oop.level.NORMAL)

    # Convert UTC datetime object to New York TimeZone
    ny = utc.astimezone(timezone('America/New_York'))
    self.oop.log.writeToLog(ny.strftime(fmtStrZone),
                           self.oop.level.DEBUG)

    # update GUI label with NY Time and Zone
    self.oop.lbl2.set(ny.strftime(fmtStrZone))|
```

Mentre stiamo stampando i valori delle diverse posizioni del fuso orario, a lungo termine, è molto più conveniente ed efficiente impostare gli orologi di debug. Non dobbiamo ingombrare il nostro codice con il vecchio stile Cprintf() dichiarazioni.



Se sei interessato a imparare come installare Eclipse con il plugin PyDev per Python, c'è un ottimo tutorial che ti farà iniziare con l'installazione di tutto il software gratuito necessario e poi ti introdurrà a PyDev all'interno di Eclipse creando un semplice programma Python funzionante : <http://www.vogella.com/tutorials/Python/article.html>

Come funziona...

Usiamo i moderni ambienti di sviluppo integrato (IDE) nel 21° secolo che sono disponibili gratuitamente per aiutarci a creare codice solido.

Questa ricetta ha mostrato come impostare gli orologi di debug, che è uno strumento fondamentale nel set di abilità di ogni sviluppatore. Passare attraverso il nostro codice anche quando non si cercano bug ci assicura di comprendere il nostro codice e può portare a migliorare il nostro codice tramite il refactoring.

Quella che segue è una citazione dal primo libro di programmazione che ho letto, *Pensando in Java*, scritto da Bruce Eckel.

"Resisti all'impulso di sbrigarti, ti rallenterà solo."

- Bruce Eckel

Quasi due decenni dopo, questo consiglio ha superato la prova del tempo.

Gli orologi di debug ci aiutano a creare un codice solido e non è una perdita di tempo.



Configurazione di diversi livelli di output di debug

In questa ricetta, configureremo diversi livelli di debug, che possiamo selezionare e modificare in fase di esecuzione. Questo ci consente di controllare quanto vogliamo approfondire il nostro codice durante il debug del nostro codice.

Creeremo due nuove classi Python e le collocheremo entrambe nello stesso modulo.

Utilizzeremo quattro diversi livelli di registrazione e scriveremo il nostro output di debug in un file di registro che creeremo. Se la log cartella non esiste, la creeremo automaticamente anche noi.

Il nome del file di registro è il nome dello script in esecuzione, che è il nostro refactoring GUI.py. Possiamo anche scegliere altri nomi per i nostri file di registro passando il percorso completo all'inizializzatore del nostro Logger classe.

Prepararsi

Continueremo a utilizzare il nostro refactoring GUI.py codice della ricetta precedente.

Come farlo...

Innanzitutto, creiamo un nuovo modulo Python in cui posizioniamo due nuove classi. La prima classe è molto semplice e definisce i livelli di registrazione. Questa è fondamentalmente un'enumerazione:

```
livello di log di classe:  
'''Definire i livelli di registrazione.'''  
SPENTO = 0  
MINIMO = 1  
NORMALE = 2  
DEBUG = 3
```

La seconda classe crea un file di registro utilizzando il percorso completo passato del nome del file e lo inserisce in a log cartella. Alla prima corsa, illog la cartella potrebbe non esistere, quindi il codice crea automaticamente la cartella:

```
importazione sistema operativo, ora  
from datetime import datetime class  
Logger:  
    ''' Crea un registro di prova e scrivici. '''  
    # -----  
    def __init__(self, fullTestName, loglevel=LogLevel.DEBUG):  
        testName = os.path.splitext(os.path.basename(fullTestName))[0] logName =  
        testName + '.log'  
  
        logsFolder = 'log'  
        se non os.path.exists(logsFolder):  
            os.makedirs(logsFolder, exist_ok = True)  
  
        self.log = os.path.join(logsFolder, logName) self.createLog()  
  
        self.loggingLevel = livello di log self.startTime =  
        time.perf_counter()  
  
    # -----  
    def createLog(self):  
        con open(self.log, mode='w', encoding='utf-8') come logFile:  
            logFile.write(self.getDateTime() +  
                         '\t\t*** Inizio test ***\n')  
        logFile.close()
```

Per scrivere nel nostro file di registro, usiamo il writeToLog() metodo. All'interno del metodo, la prima cosa che facciamo è controllare se il messaggio ha un livello di registrazione superiore al limite su cui abbiamo impostato l'output di registrazione desiderato. Se il messaggio ha un livello inferiore, lo scartiamo e torniamo immediatamente dal metodo.

Se il messaggio ha il livello di registrazione che vogliamo visualizzare, controlliamo se inizia con un carattere di nuova riga e, in caso affermativo, scartiamo la nuova riga affettando il metodo a partire dall'indice 1, utilizzando l'operatore slice di Python (msg = msg[1:]).

Quindi scriviamo una riga nel nostro file di registro che consiste nel timestamp della data corrente, due spazi di tabulazione, il nostro messaggio e termina con un carattere di nuova riga:

```
# -----
def writeToLog(self, msg="", loglevel=LogLevel.DEBUG):
    # controlla quanto viene registrato
    if loglevel > self.loggingLevel:
        ritorno

    # apre il file di registro in modalità di aggiunta
    con open(self.log, mode='a', encoding='utf-8') come logFile:
        msg = str(msg)
        if msg.startswith('\n'):
            messaggio = messaggio[1:]
        logFile.write(self.getDateTime() + '\t\t' + msg + '\n')

    logFile.close()
```

Ora possiamo importare il nostro nuovo modulo Python e, all'interno del _dentro_ sezione del nostro codice GUI, può creare un'istanza del Logger classe:

```
dal percorso di importazione del sistema operativo
da Ch08_Code.Logger import Classe Logger
OOP():
    def __init__(self):
        # crea un'istanza del logger
        fullPath = percorso.realpath(__file__) self.log =
        Logger(fullPath)
        print(self.log)
```

Stiamo recuperando il percorso completo del nostro script GUI in esecuzione tramite percorso.percorso reale(__file__) e passando questo nell'inizializzatore di Logger classe. Se la log la cartella non esiste, verrà creata automaticamente dal nostro codice Python.

Questo crea i seguenti risultati:

Callbacks_Refactored.py con print(self.log) non commentato

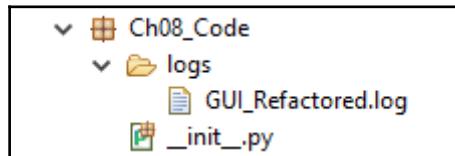
The screenshot shows a code editor window with the following Python code:

```
# create Logger instance
fullPath = path.realpath(__file__)
self.log = Logger(fullPath)
print(self.log)
```

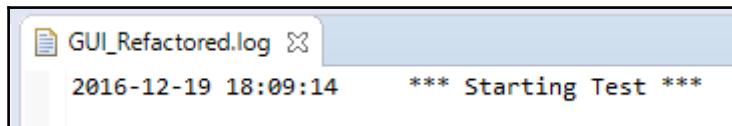
Below the code editor is a terminal window titled "Console". The output shows:

```
<terminated> C:\Eclipse_NEON_workspace\2nd Edition Python GUI Programming
<Ch08_Code.Logger object at 0x000001FFAD0C5CF8>
```

Lo screenshot precedente mostra che abbiamo creato un'istanza del nostro nuovo Logger class e lo screenshot seguente mostra che entrambi i log cartella e il registro sono stati creati:



Quando apriamo il registro, possiamo vedere che la data e l'ora correnti e una stringa predefinita sono state scritte nel registro:



Come funziona...

In questa ricetta, abbiamo creato la nostra classe di registrazione. Mentre Python viene fornito con aRegistrazione modulo, è molto facile creare uno nostro, che ci dà il controllo assoluto sul nostro formato di registrazione. Questo è molto utile quando combiniamo il nostro output di registrazione con MS Excel o il Matplotlib abbiamo esplorato nelle ricette di un capitolo precedente.

Nella prossima ricetta, useremo il built-in di Python __principale__ funzionalità per utilizzare i quattro diversi livelli di registrazione che abbiamo appena creato.

Creazione di codice di autotest utilizzando la sezione `_main_` di Python

Python è dotato di una funzionalità molto interessante che consente a ciascun modulo di eseguire l'autotest. L'utilizzo di questa funzione è un ottimo modo per assicurarsi che le modifiche al nostro codice non infrangano il codice esistente e, inoltre, il `_principale_` la sezione di autotest può servire come documentazione per il funzionamento di ciascun modulo.



Dopo alcuni mesi o anni, a volte dimentichiamo cosa sta facendo il nostro codice, quindi avere una spiegazione scritta nel codice stesso è davvero di grande beneficio.

È una buona idea aggiungere sempre una sezione di autotest a ogni modulo Python, quando possibile. A volte non è possibile ma, nella maggior parte dei moduli, è possibile farlo.

Prepararsi

Estenderemo la ricetta precedente, quindi per capire cosa sta facendo il codice in questa ricetta, dobbiamo prima leggere e comprendere il codice della ricetta precedente.

Come farlo...

Per prima cosa, esploreremo la potenza di Python `_principale_` sezione di autotest aggiungendo questa sezione di autotest al nostro `LanguageResources.py` modulo. Ogni volta che eseguiamo un modulo che ha questa sezione di autotest situata nella parte inferiore del modulo, quando il modulo viene eseguito da solo, verrà eseguito questo codice.

Quando il modulo viene importato e utilizzato da altri moduli, il codice nel `_principale_` la sezione di autotest non verrà eseguita.

Questo è il codice che viene mostrato anche nello screenshot che segue:

```
if __name__ == '__main__':
    lingua = 'it'
    inst = I18N(lingua)
    print(inst.titolo)

    lingua = 'de'
    inst = I18N(lingua)
    print(inst.titolo)
```

Dopo aver aggiunto la sezione di autotest, ora possiamo eseguire questo modulo da solo e crea un output utile mentre allo stesso tempo ci mostra che il nostro codice funziona come previsto:

LanguageResources.py

The screenshot shows the Eclipse IDE interface. In the top editor window, there is Python code. Below it, in the bottom editor window, is the 'Console' tab showing the execution output.

```
if __name__ == '__main__':
    language = 'en'
    inst = I18N(language)
    print(inst.title)

    language = 'de'
    inst = I18N(language)
    print(inst.title)
```

Console output:

```
<terminated> C:\Eclipse_NEON_workspace\2nd Edition Python GUI Programming
Python Graphical User Interface
Python Grafische Benutzeroberfläche
```

Prima passiamo l'inglese come lingua da visualizzare nella nostra GUI e poi passeremo il tedesco come lingua che verrà visualizzata dalla nostra GUI.

Stampiamo il titolo della nostra GUI per mostrare che il nostro modulo Python funziona come volevamo che funzionasse.



Il passaggio successivo consiste nell'utilizzare le nostre capacità di registrazione, che abbiamo creato nella ricetta precedente.

Lo facciamo aggiungendo prima un `__principale__` sezione di autotest al nostro refactoring `GUI_Refactored.py` modulo e quindi verifichiamo di aver creato un'istanza del nostro Logger classe:

GUI_Refactored.py con print(oop.log) non commentato

The screenshot shows the Eclipse IDE interface. In the top-left, there's a code editor window containing Python code. Below it is a terminal window titled 'Console'.

```
#=====
if __name__ == '__main__':
# =====
# Start GUI
# =====
oop = OOP()
print(oop.log)
oop.win.mainloop()
```

Console output:

```
<terminated> C:\Eclipse_NEON_workspace\2nd Edition Python GUI Progr
<Ch08_Code.Logger.Logger object at 0x0000020CE0386CF8>
```

Successivamente scriviamo nel nostro file di registro usando il comando mostrato. Abbiamo progettato il nostro livello di registrazione in modo predefinito per registrare ogni messaggio, che è il DEBUG livello, e per questo non dobbiamo cambiare nulla. Passiamo semplicemente il messaggio per essere loggato alwriteToLog metodo:

```
if __name__ == '__main__':
# =====
# Avvia GUI
# =====
ops = OOP()
print(oop.log)
oop.log.writeToLog('Messaggio di prova')
oop.win.mainloop()
```

Questo viene scritto nel nostro file di registro, come si può vedere nella seguente schermata del registro:

The screenshot shows a log file titled 'GUI_Refactored.log'. It contains the following entries:

```
2016-12-19 18:26:35    *** Starting Test ***
2016-12-19 18:26:35    Test message
```

Ora possiamo controllare la registrazione aggiungendo livelli di registrazione alle nostre istruzioni di registrazione e impostando il livello che desideriamo emettere. Aggiungiamo questa capacità al nostro **New York** metodo di callback del pulsante nel `Callbacks_Refactored.py` modulo, che è il `getDateTime` metodo.

Cambiamo il precedente Stampa dichiarazioni a log istruzioni utilizzando diversi livelli di debug. Nel `GUI_Refactored.py`, importiamo entrambe le nuove classi dal nostro `Logger` modulo:

```
da Ch08_Code.Logger import Logger, LogLevel
```

Successivamente, creiamo istanze locali di tali classi:

```
# crea un'istanza del logger
fullPath = percorso.realpath(__file__)
self.log = Logger(fullPath)

# crea un'istanza a livello di registro
self.level = LogLevel()
```

Poiché stiamo passando un'istanza della classe `GUI` a `GUI` `Callbacks_Refactored.py` inizializzatore, possiamo usare i vincoli del livello di registrazione in base al `LogLevel` classe che abbiamo creato:

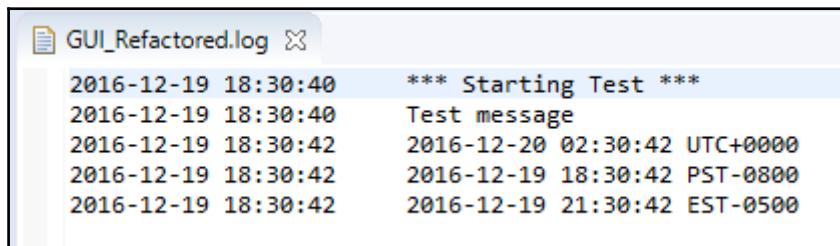
```
# Formatta l'ora locale degli Stati Uniti con le informazioni sul fuso orario
def getDateTime(self):
    fmtStrZone = "%Y-%m-%d %H:%M:%S %Z%z"
    # Ottieni tempo universale coordinato
    utc = datetime.now(timezone('UTC'))
    self.oop.log.writeToLog(utc.strftime(fmtStrZone),
                           self.oop.level.MINIMUM)

    # Converti oggetto datetime UTC in Los Angeles TimeZone
    la = utc.astimezone(timezone('America/Los_Angeles'))
    self.oop.log.writeToLog(la.strftime(fmtStrZone),
                           self.oop.level.NORMAL)

    # Converti oggetto datetime UTC in New York TimeZone
    ny = utc.astimezone(timezone('America/New_York'))
    self.oop.log.writeToLog(ny.strftime(fmtStrZone),
                           self.oop.level.DEBUG)

    # aggiorna l'etichetta della GUI con l'ora e il fuso di New York
    self.oop.lbl2.set(ny.strftime(fmtStrZone))
```

Quando ora clicchiamo sul nostro **New York** pulsante, a seconda del livello di registrazione selezionato, otteniamo un output diverso scritto nel nostro file di registro. Il livello di registrazione predefinito è DEBUG, il che significa che tutto viene scritto nel nostro log:



Timestamp	Message
2016-12-19 18:30:40	*** Starting Test ***
2016-12-19 18:30:40	Test message
2016-12-19 18:30:42	2016-12-20 02:30:42 UTC+0000
2016-12-19 18:30:42	2016-12-19 18:30:42 PST-0800
2016-12-19 18:30:42	2016-12-19 21:30:42 EST-0500

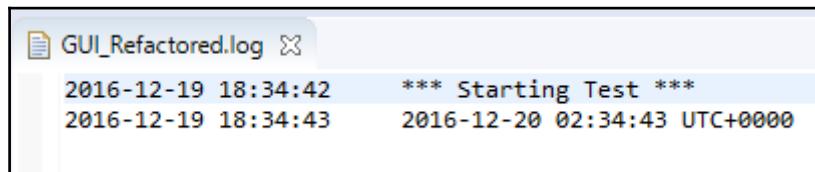
Quando cambiamo il livello di registrazione, controlliamo ciò che viene scritto nel nostro registro. Lo facciamo chiamando il `setLoggingLevel` metodo del `Logger` classe:

```
# -----
def setLoggingLevel(self, level):
    '''cambia il livello di registrazione nel mezzo di un test.'''
    self.loggingLevel = level
```

Nel `__principale__` sezione della nostra GUI, cambiamo il livello di registrazione in MINIMO, che si traduce in un output ridotto scritto nel nostro file di registro:

```
if __name__ == '__main__':
# =====
# Avvia GUI
# =====
ops = OOP()
oop.log.setLoggingLevel(oop.level.MINIMUM)
oop.log.writeToLog('Messaggio di prova')
oop.win.mainloop()
```

Ora, il nostro file di registro non mostra più il Messaggio di prova e mostra solo i messaggi che soddisfano il livello di registrazione impostato:



Timestamp	Message
2016-12-19 18:34:42	*** Starting Test ***
2016-12-19 18:34:43	2016-12-20 02:34:43 UTC+0000

Come funziona...

In questa ricetta, abbiamo fatto buon uso del built-in di Python `_principale_` sezione di autodiagnosi. Abbiamo introdotto il nostro file di registrazione e, allo stesso tempo, abbiamo imparato a creare diversi livelli di registrazione. In questo modo, abbiamo il pieno controllo su ciò che viene scritto nei nostri file di registro.

Creazione di GUI robuste utilizzando i test di unità

Python viene fornito con un framework di test delle unità integrato e, in questa ricetta, inizieremo a utilizzare questo framework per testare il nostro codice GUI Python.

Prima di iniziare a scrivere i test unitari, vogliamo progettare la nostra strategia di test. Potremmo facilmente mescolare gli unit test con il codice che stanno testando, ma una strategia migliore consiste nel separare il codice dell'applicazione dal codice del test unitario.



PyUnit è stato progettato secondo i principi di tutti gli altri framework di test xUnit.

Prepararsi

Verificheremo la GUI internazionalizzata che abbiamo creato in precedenza in questo capitolo.

Come farlo...

Per utilizzare il framework di test delle unità integrato di Python, dobbiamo importare il Python `unittest` modulo. Creiamo un nuovo modulo e chiamiamolo `UnitTest.py`.

Per prima cosa importiamo il `unittest` modulo, quindi creiamo la nostra classe e, all'interno di questa classe, ereditiamo ed estendiamo la `unittest.TestCase` classe.

Il codice più semplice per farlo è il seguente:

```
import unittest

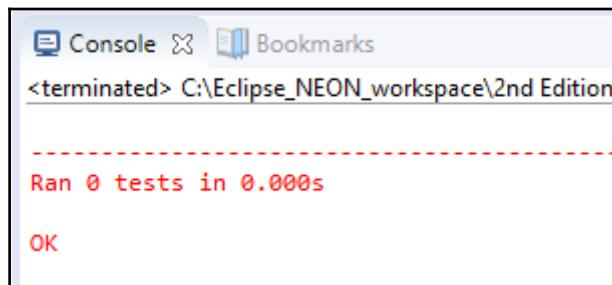
class GuiUnitTests(unittest.TestCase):
    passaggio

    if __name__ == '__main__':
```

```
unittest.main()
```

Il codice non sta ancora facendo molto, ma quando lo eseguiamo, non riceviamo errori, il che è un buon segno:

UnitTestMinimum.py



In realtà otteniamo un output scritto sulla console che afferma che abbiamo eseguito con successo zero test ...

Quell'output è un po' fuorviante, poiché tutto ciò che abbiamo fatto finora è creare una classe che non contenga metodi di test effettivi.

Aggiungiamo metodi di test che eseguono l'effettivo test unitario seguendo la denominazione predefinita per tutti i metodi di test per iniziare con la parola test. Questa è un'opzione che può essere modificata, ma è molto più semplice e chiara seguire questa convenzione di denominazione.

Aggiungiamo un metodo di prova che testerà il titolo della nostra GUI. Questo verificherà che, passando gli argomenti previsti, otteniamo il risultato atteso:

```
import unittest
da Ch08_Code.LanguageResources import I18N

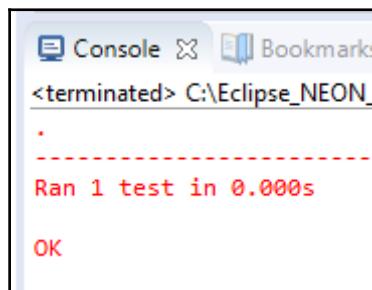
class GuiUnitTests(unittest.TestCase):

    def test_TitleIsEnglish(self):
        i18n = I18N('en')
        self.assertEqual(i18n.title, "Interfaccia utente grafica Python")
```

Stiamo importando il nostro I18N classe dal nostro Risorse.py modulo, passando l'inglese come lingua da visualizzare nella nostra GUI. Poiché questo è il nostro primo test unitario, stamperemo anche il risultato del titolo, solo per assicurarci di sapere cosa stiamo ricevendo. Usiamo poi il unittest assertEquals metodo per verificare che il nostro titolo sia corretto.

L'esecuzione di questo codice ci dà un **OK**, il che significa che il test dell'unità ha superato:

UnitTests_One.py



Il test dell'unità viene eseguito e ha esito positivo, come indicato da un punto e dalla parola **OK**. Se avesse fallito o avesse ricevuto un errore, non avremmo ottenuto il punto ma un **F** o **E** come uscita.

Ora possiamo eseguire lo stesso controllo di unit test automatizzato verificando il titolo per la versione tedesca della nostra GUI. Semplicemente copiamo, incolliamo e modifichiamo il nostro codice:

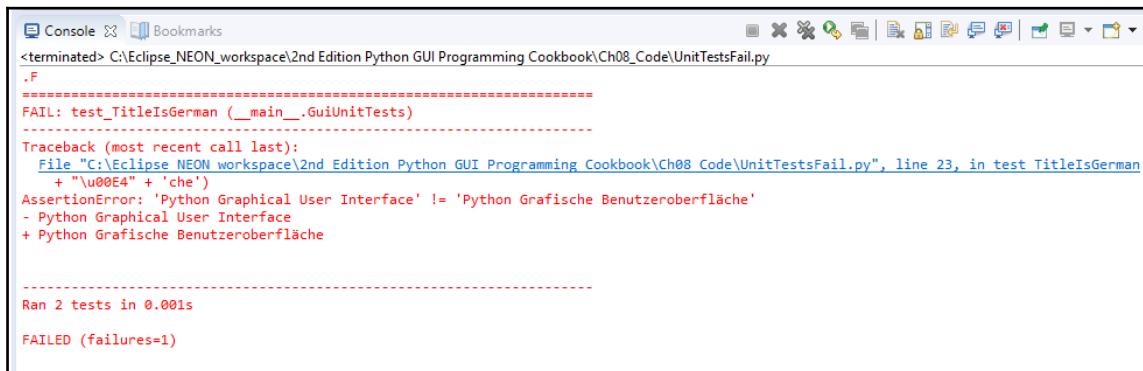
```
import unittest
da Ch08_Code.LanguageResources import I18N

class GuiUnitTests(unittest.TestCase):

    def test_TitleIsEnglish(self):
        i18n = I18N('en')
        self.assertEqual(i18n.title, "Interfaccia utente grafica Python")

    def test_TitleIsGerman(self):
        i18n = I18N('de')
        self.assertEqual(i18n.title,
                        'Python Grafische Benutzeroberfl' + "u00E4" + 'che')
```

Ora, testiamo il nostro titolo GUI internazionalizzato in due lingue e otteniamo il seguente risultato durante l'esecuzione del codice:



The screenshot shows the Eclipse IDE's Console view with the following output:

```
<terminated> C:\Eclipse_NEON_workspace\2nd Edition Python GUI Programming Cookbook\Ch08_Code\UnitTestsFail.py
.F
=====
FAIL: test_TitleIsGerman (_main_.GuiUnitTests)
-----
Traceback (most recent call last):
  File "C:\Eclipse_NEON_workspace\2nd Edition Python GUI Programming Cookbook\Ch08_Code\UnitTestsFail.py", line 23, in test_TitleIsGerman
    + "\u00e4" + 'che')
AssertionError: 'Python Graphical User Interface' != 'Python Grafische Benutzeroberfl\u00e4che'
- Python Graphical User Interface
+ Python Grafische Benutzeroberfl\u00e4che

-----
Ran 2 tests in 0.001s
FAILED (failures=1)
```

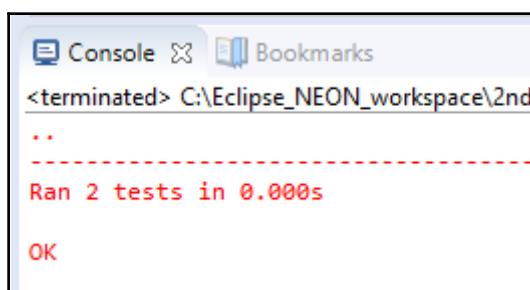
Abbiamo eseguito due unit test ma, invece di un OK, abbiamo avuto un fallimento. Quello che è successo? Nostroasserzione fallito per la versione tedesca della nostra GUI...

Durante il debug del nostro codice, si scopre che nell'approccio copia, incolla e modifica del nostro codice di test dell'unità, abbiamo dimenticato di passare il tedesco come lingua. Possiamo facilmente risolvere questo problema:

```
def test_TitleIsGerman(self):
    # i18n = I18N('en')                      # <= Bug nel test dell'unità
    i18n = I18N('de')
    self.assertEqual(i18n.title,
                    'Python Grafische Benutzeroberfl\u00e4che' + "\u00e4" + 'che')
```

Quando eseguiamo nuovamente i nostri test unitari, otteniamo il risultato atteso di tutti i nostri test passati:

UnitTestsFail.py con errore corretto



The screenshot shows the Eclipse IDE's Console view with the following output:

```
<terminated> C:\Eclipse_NEON_workspace\2nd
...
Ran 2 tests in 0.000s
OK
```



Il codice di unit test è codice e può avere anche bug.

Sebbene lo scopo di scrivere unit test sia realmente quello di testare il codice della nostra applicazione, dobbiamo assicurarci che i nostri test siano scritti correttamente. Un approccio da **Sviluppo guidato dai test (TDD)** la metodologia potrebbe aiutarci.



In TDD, sviluppiamo gli unit test prima di scrivere effettivamente il codice dell'applicazione.

Ora, se un test passa per un metodo che nemmeno esiste, qualcosa non va. Il passaggio successivo consiste nel creare il metodo inesistente e assicurarsi che non funzionerà.

Successivamente, possiamo scrivere la quantità minima di codice necessaria per far passare il test dell'unità.

Come funziona...

In questa ricetta abbiamo iniziato a testare la nostra GUI Python, scrivendo unit test in Python. Abbiamo visto che il codice di test dell'unità Python è solo codice e può contenere errori che devono essere corretti. Nella prossima ricetta, estenderemo il codice di questa ricetta e utilizzeremo il corridore di test di unità grafico fornito con il plug-in PyDev per l'IDE Eclipse.

Come scrivere unit test utilizzando l'IDE Eclipse PyDev

Nella ricetta precedente, abbiamo iniziato a utilizzare le capacità di test delle unità di Python e, in questa ricetta, garantiremo la qualità del nostro codice GUI utilizzando ulteriormente questa funzionalità.

Verificheremo la nostra GUI per assicurarci che le stringhe internazionalizzate visualizzate dalla nostra GUI siano come previsto.

Nella ricetta precedente, abbiamo riscontrato alcuni bug nel nostro codice di unit test ma, in genere, i nostri unit test troveranno bug di regressione causati dalla modifica del codice dell'applicazione esistente, non del codice di unit test. Dopo aver verificato che il nostro codice di test dell'unità è corretto, di solito non lo modifichiamo.



I nostri test di unità servono anche come documentazione di ciò che ci aspettiamo che faccia il nostro codice.

Per impostazione predefinita, gli unit test di Python vengono eseguiti con un runner test di unità testuale e possiamo eseguirlo nel plug-in PyDev dall'interno dell'IDE Eclipse. Possiamo anche eseguire gli stessi unit test da una finestra della console.

Oltre al text runner in questa ricetta, esploreremo la funzione di test dell'unità grafica di PyDev, che può essere utilizzata dall'interno dell'IDE Eclipse.

Prepararsi

Estenderemo la ricetta precedente in cui abbiamo iniziato a utilizzare i test di unità Python.

Come farlo...

Il framework di test delle unità Python viene fornito con ciò che viene chiamato **Infissi**.

Fare riferimento ai seguenti URL per una descrizione di cos'è un dispositivo di prova:

- <https://docs.python.org/3.6/library/unittest.html>
- en.wikipedia.org/wiki/Test_fixture
- [http://www.boost.org/doc/libs/1_51_0/libs/test/doc/html/utf/user-guide\(fixture.html](http://www.boost.org/doc/libs/1_51_0/libs/test/doc/html/utf/user-guide(fixture.html)

Ciò significa che possiamo creare impostare() e demolire() metodi di test unitari in modo che impostare() viene chiamato all'inizio prima che venga eseguito ogni singolo test, e alla fine di ogni singolo test unitario, il demolire() viene chiamato il metodo.



Questa capacità del dispositivo ci fornisce un ambiente molto controllato in cui possiamo eseguire i nostri test unitari. È simile all'utilizzo di pre e post condizioni.

Configuriamo il nostro ambiente di test delle unità. Creeremo una nuova classe di test che si concentra sulla suddetta correttezza del codice:

```
import unittest  
da Ch08_Code.LanguageResources import I18N
```

```
da Ch08_Code.GUI_Refactored importa OOP come GUI

class GuiUnitTests(unittest.TestCase):
    def test_TitleIsEnglish(self):
        i18n = I18N('en')
        self.assertEqual(i18n.title, "Interfaccia utente grafica Python")

    def test_TitleIsGerman(self):
        # i18n = I18N('en') # <= Bug in Unit Test
        i18n = I18N('de')
        self.assertEqual(i18n.title,
                        'Python Grafische Benutzeroberfl\u00e4che')

class WidgetsTestsEnglish(unittest.TestCase):
    def setUp(self):
        self.gui = GUI('en')

    def tearDown(self):
        self.gui = Nessuno

    def test_WidgetLabels(self):
        self.assertEqual(self.gui.i18n.file, "File")
        self.assertEqual(self.gui.i18n.mgrFiles, 'Gestisci file')
        self.assertEqual(self.gui.i18n.browseTo, "Browse to File...")

# ===== if
if __name__ == '__main__':
    unittest.main()
```



unittest.main() esegue qualsiasi metodo che inizi con il test del prefisso, indipendentemente dal numero di classi che creiamo all'interno di un dato modulo Python.

Questo dà il seguente output:

UnitTestEnglish.py

A screenshot of a terminal window titled 'Console'. The window shows the command 'C:\Eclipse_NEON_workspace\2nd Edition' being run. It then displays three red dots indicating a loop or continuation, followed by the text 'Ran 3 tests in 0.132s' in red, which is a standard message from the unittest module. At the bottom, the word 'OK' is visible, indicating the test run was successful.

```
Console Bookmarks
<terminated> C:\Eclipse_NEON_workspace\2nd Edition
...
Ran 3 tests in 0.132s
OK
```

Il codice di test delle unità sopra mostra che possiamo creare diverse classi di test delle unità e possono essere eseguite tutte nello stesso modulo chiamando `unittest.main()`.

Mostra anche che impostare() il metodo non conta come test nell'output del rapporto di test dell'unità (il conteggio dei test è 3) mentre, allo stesso tempo, ha svolto il lavoro previsto poiché ora possiamo accedere alla nostra variabile di istanza di classe `self.gui` dall'interno del metodo di unit test.

Siamo interessati a testare la correttezza di tutte le nostre etichette e, soprattutto, a rilevare i bug quando apportiamo modifiche al nostro codice.

Se abbiamo copiato e incollato stringhe dal nostro codice dell'applicazione al codice di test, catturerà eventuali modifiche indesiderate con il clic di un pulsante del framework di test delle unità.

Vogliamo anche testarlo invocando uno dei nostri Pulsante radio widget in qualsiasi lingua si traduce in EtichettaFrame aggiungendo testo in fase di aggiornamento. Per testarlo automaticamente, dobbiamo fare due cose.

Per prima cosa, dobbiamo recuperare il valore di EtichettaFrame widget e assegna il valore a una variabile che chiamiamo `labelFrameText`. Dobbiamo utilizzare la seguente sintassi perché le proprietà di questo widget vengono passate e recuperate tramite un tipo di dati del dizionario:

```
self.gui.widgetFrame['testo']
```

Ora possiamo verificare il testo predefinito e quindi le versioni internazionalizzate dopo aver fatto clic su uno dei pulsanti di opzione a livello di codice:

```
class WidgetsTestsGerman(unittest.TestCase):
    def setUp(self):
        self.gui = GUI('de')

    def test_WidgetLabels(self):
        self.assertEqual(self.gui.i18n.file, "Datei") self.assertEqual(self.gui.i18n.mgrFiles, ' Dateien
Organisieren ') self.assertEqual(self.gui.i18n.browseTo, "Waehle eine Datei. ...")

    def test_LabelFrameText(self):
        labelFrameText = self.gui.widgetFrame['text']
        self.assertEqual(labelFrameText, " Widgets Rahmen ")
        self.gui.radVar.set(1)
        self.gui.callBacks.radCall()
        labelFrameText = self.gui.widgetFrame['text'] self.assertEqual(labelFrameText, "
Widget Rahmen in Gold")
```

Dopo aver verificato l'impostazione predefinita etichettaFrameTesto, impostiamo a livello di codice il pulsante di opzione sull'indice 1 e quindi invochiamo il metodo di callback del pulsante di opzione:

```
self.gui.radVar.set(1)  
self.gui.callBacks.radCall()
```

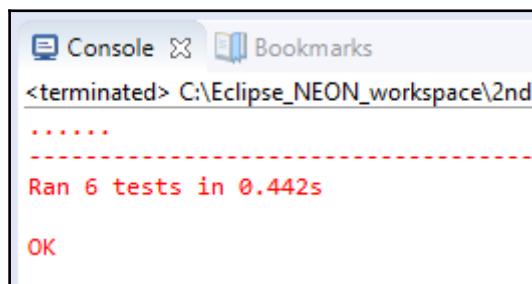


Questa è fondamentalmente la stessa azione del clic sul pulsante di opzione nella GUI, ma eseguiamo questo evento di clic del pulsante tramite codice negli unit test.

Quindi verifichiamo che il nostro testo in EtichettaFrame widget è cambiato come previsto.

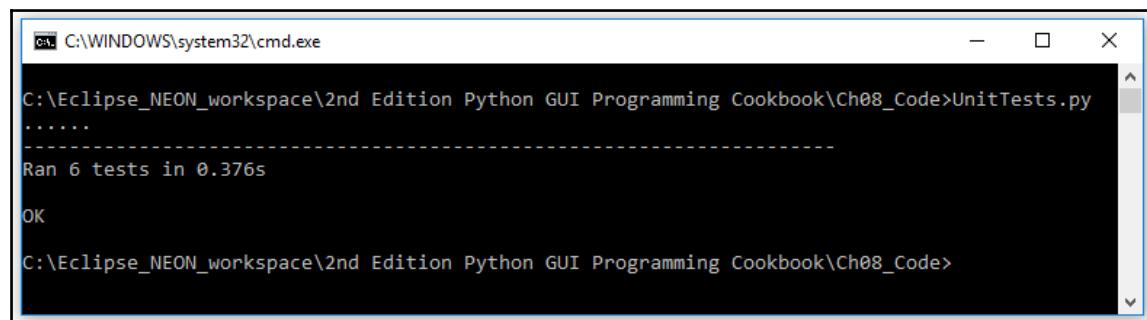
Quando eseguiamo gli unit test da Eclipse con il plugin Python PyDev, otteniamo il seguente output scritto nella console di Eclipse:

UnitTest.py



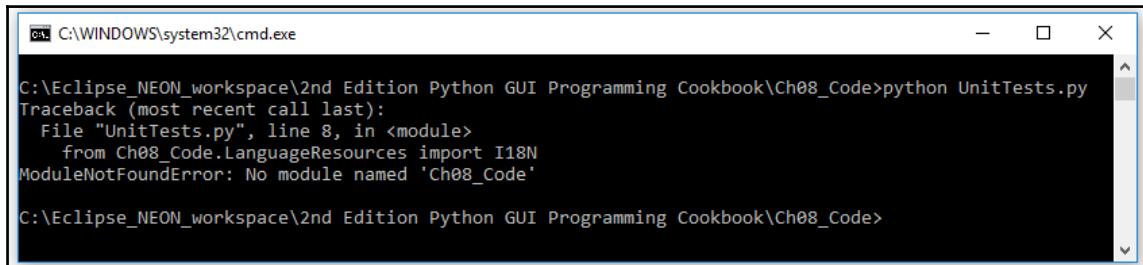
A screenshot of the Eclipse IDE's Console view. The title bar says "Console". The content shows the output of a test run: "<terminated> C:\Eclipse_NEON_workspace\2nd", followed by several dots, then a red dashed line, and the text "Ran 6 tests in 0.442s" in red, and finally "OK" in black.

Eseguito da un prompt dei comandi, otteniamo un output simile una volta che navighiamo nella cartella in cui risiede il nostro codice:



A screenshot of a Windows Command Prompt window titled "cmd.exe". The path is "C:\Eclipse_NEON_workspace\2nd Edition Python GUI Programming Cookbook\Ch08_Code>". The output shows the command "UnitTest.py", followed by several dots, then a red dashed line, and the text "Ran 6 tests in 0.376s" in red, and finally "OK" in black.

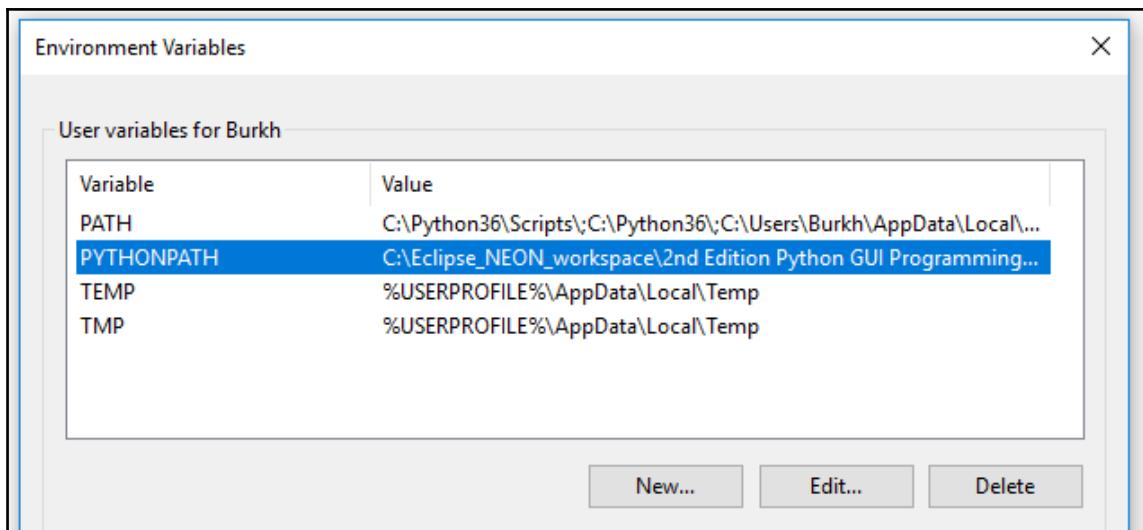
Se ottieni un Errore modulo non trovato, aggiungi semplicemente la directory in cui risiede il tuo codice Python alla variabile ambientale PYTHONPATH di Windows, come mostrato nelle schermate seguenti:



C:\WINDOWS\system32\cmd.exe

```
C:\Eclipse_NEON_workspace\2nd Edition Python GUI Programming Cookbook\Ch08_Code>python UnitTests.py
Traceback (most recent call last):
  File "UnitTests.py", line 8, in <module>
    from Ch08_Code.LanguageResources import I18N
ModuleNotFoundError: No module named 'Ch08_Code'

C:\Eclipse_NEON_workspace\2nd Edition Python GUI Programming Cookbook\Ch08_Code>
```

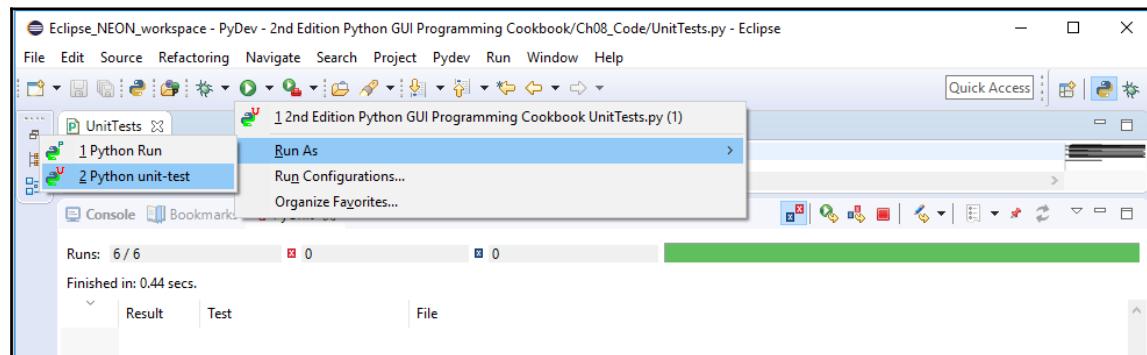


Per esempio, C:\Eclipse_NEON_workspace\2nd Edition Python GUI Programming Cookbook:

Name	Date modified	Type	Size
.settings	12/18/2016 4:27 PM	File folder	
Ch01_Code	11/24/2016 11:31 AM	File folder	
Ch02_Code	11/26/2016 8:34 PM	File folder	
Ch03_Code	11/29/2016 10:41 PM	File folder	
Ch04_Code	12/4/2016 12:54 PM	File folder	
Ch05_Code	12/6/2016 11:02 PM	File folder	
Ch06_Code	12/12/2016 1:39 AM	File folder	
Ch07_Code	12/14/2016 8:40 PM	File folder	
Ch08_Code	12/19/2016 7:37 PM	File folder	

Questo riconoscerà il Ch08_Codice folder come pacchetto Python e il codice verrà eseguito.

Usando Eclipse, possiamo anche scegliere di eseguire i nostri unit test, non come un semplice script Python, ma come a *Python unit test* script, che ci fornisce un output colorato invece del mondo in bianco e nero del prompt del DOS:



La barra dei risultati del test dell'unità è verde, il che significa che tutti i nostri test dell'unità sono stati superati. Lo screenshot precedente mostra anche che il test runner della GUI è più lento del test runner testuale: 0,44 secondi rispetto a 0,376 secondi in Eclipse.

Come funziona...

Abbiamo esteso il nostro codice di unit test testando etichette, invocando a livello di codice a Pulsante radio, e quindi verificando nei nostri unit test che il corrispondente testo proprietà del EtichettaFrame widget è cambiato come previsto. Abbiamo testato due lingue diverse. Siamo quindi passati all'utilizzo dell'unità di test runner grafico Eclipse/PyDev integrato.

9

Estendere la nostra GUI con il wxPython Library

In questo capitolo, miglioreremo la nostra GUI Python usando il wxPython biblioteca.
Tratteremo le seguenti ricette:

- Installazione del wxPython libreria
- Creazione della nostra GUI in wxPython
- Aggiunta rapida di controlli utilizzando wxPython
- Cercando di incorporare un'app wxPython principale in un'app tkinter
- principale Provando a incorporare il nostro codice GUI tkinter in
- wxPython Utilizzo di Python per controllare due diversi framework GUI
- Comunicare tra due GUI collegate

introduzione

In questo capitolo, introdurremo un altro toolkit della GUI Python che attualmente non viene fornito con Python. È chiamato wxPython.

Esistono due versioni di questa libreria. L'originale si chiama Classic, mentre il più recente è chiamato con il nome in codice del progetto di sviluppo, che è Phoenix.

In questo libro, programmiamo esclusivamente utilizzando Python 3.6 e versioni successive, e poiché il nuovo progetto Phoenix mira a supportare Python 3.6 e versioni successive, questa è la versione di wxPython che useremo in questo capitolo.

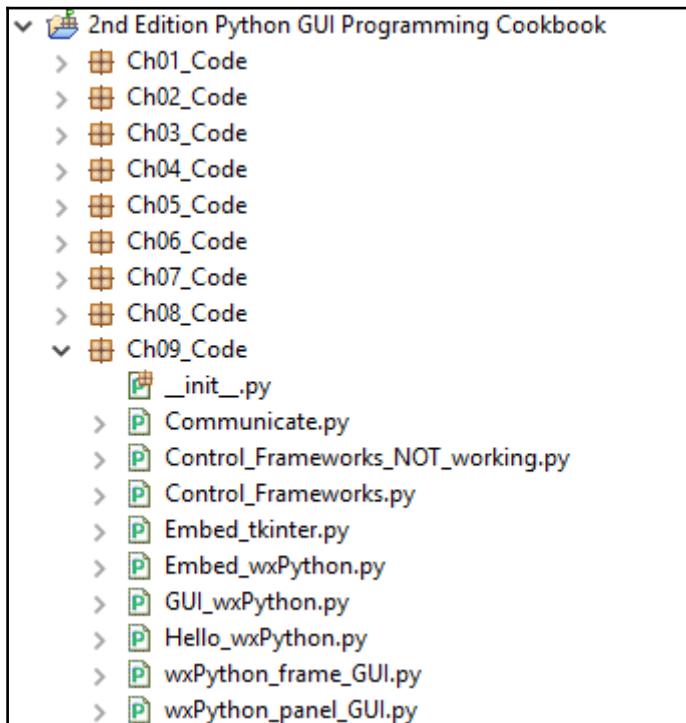
Per prima cosa, creeremo una semplice GUI wxPython, e poi proveremo a connettere entrambe le GUI basate su tkinter che abbiamo sviluppato in questo libro con il nuovo wxPython biblioteca.



wxPython è un'associazione Python a wxWidgets. La w in wxPython sta per il sistema operativo Windows e la x sta per i sistemi operativi basati su Unix, come Linux e OS X di Apple (ora ribattezzato Mac OS).

Se le cose non dovessero funzionare utilizzando questi due toolkit GUI all'unisono, cercheremo di utilizzare Python per risolvere eventuali problemi e, se necessario, utilizzeremo **Comunicazione tra processi (IPC)** all'interno di Python per assicurarci che il nostro codice Python funzioni come vogliamo che funziona.

Ecco la panoramica dei moduli Python per questo capitolo:



Installazione della libreria wxPython

La libreria wxPython non viene fornita con Python, quindi per utilizzarla dobbiamo prima installarla.

Questa ricetta ci mostrerà dove e come trovare la versione giusta da installare per abbinare sia la versione installata di Python che il sistema operativo che stiamo utilizzando.



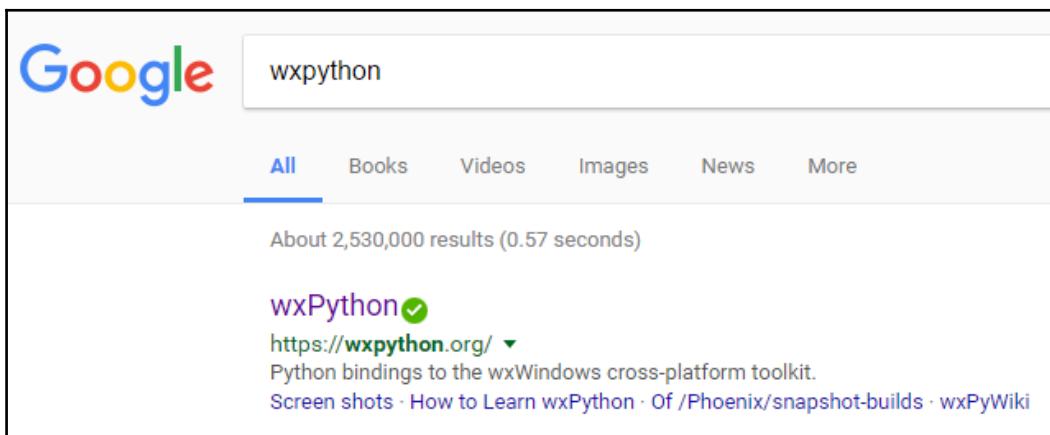
La libreria di terze parti wxPython è in circolazione da più di 18 anni, il che indica che si tratta di una libreria robusta.

Prepararsi

Per utilizzare wxPython con Python 3.6 e versioni successive, dobbiamo installare wxPython **Fenice** versione.

Come farlo...

Durante la ricerca online di wxPython, probabilmente troverai il sito Web ufficiale all'indirizzo www.wxpython.org:

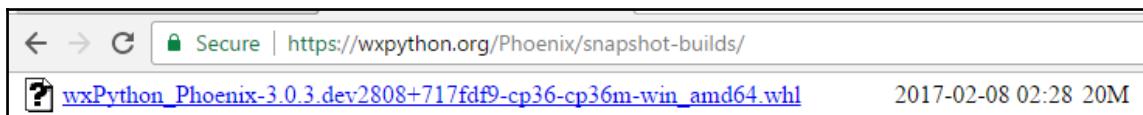


Se fai clic sul link di download per MS Windows, vedrai diversi programmi di installazione di Windows, tutti solo per Python 2.x:

The screenshot shows a web browser window with the URL <https://wxpython.org/download.php#msw>. On the left, there's a sidebar with links like 'Screen shots', 'Learning wxPython', 'Recent Changes', etc. Below it is a 'Download' section with links for 'MS Windows', 'Mac OSX', 'Linux', 'Source', and 'Build instructions'. The main content area is titled 'Windows Binaries' and contains text about choosing the right installer based on Python version and system architecture. It lists several download links: [wxPython3.0-win32-py26](#) (32-bit Python 2.6), [wxPython3.0-win64-py26](#) (64-bit Python 2.6), [wxPython3.0-win32-py27](#) (32-bit Python 2.7), and [wxPython3.0-win64-py27](#) (64-bit Python 2.7). Below this is a section titled 'wxPython Demo for Windows' with a link to [wxPython3.0-win32-docs-demos.exe](#).

Per usare wxPython con Python 3.6, dobbiamo installare la libreria wxPython/Phoenix. Possiamo trovare il programma di installazione nelle build snapshot collegamento: <http://wxpython.org/Phoenix/snapshot-builds/>

Da qui, possiamo selezionare la versione wxPython/Phoenix che corrisponde sia alle nostre versioni di Python che al nostro sistema operativo. Sto usando Python 3.6 in esecuzione su un sistema operativo Windows 10 a 64 bit.

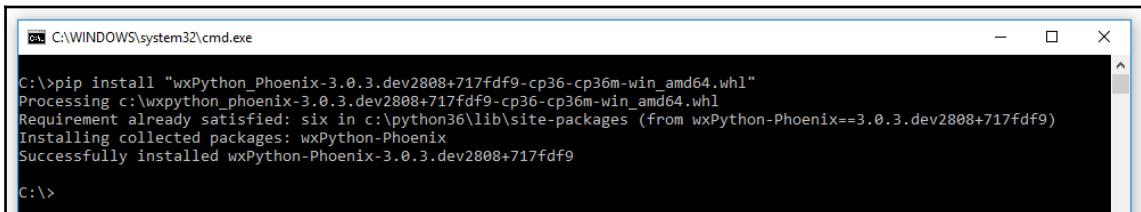


The screenshot shows a web browser window with the URL <https://wxpython.org/Phoenix/snapshot-builds/>. It displays a single download link: [wxPython_Phoenix-3.0.3.dev2808+717fdf9-cp36-cp36m-win_amd64.whl](#). Below the link, the page shows the last modified date as '2017-02-08 02:28' and the file size as '20M'. There is also a small 'Secure' icon and a question mark icon next to the file name.

La ruota di Python (.whl) pacchetto di installazione ha uno schema di numerazione.

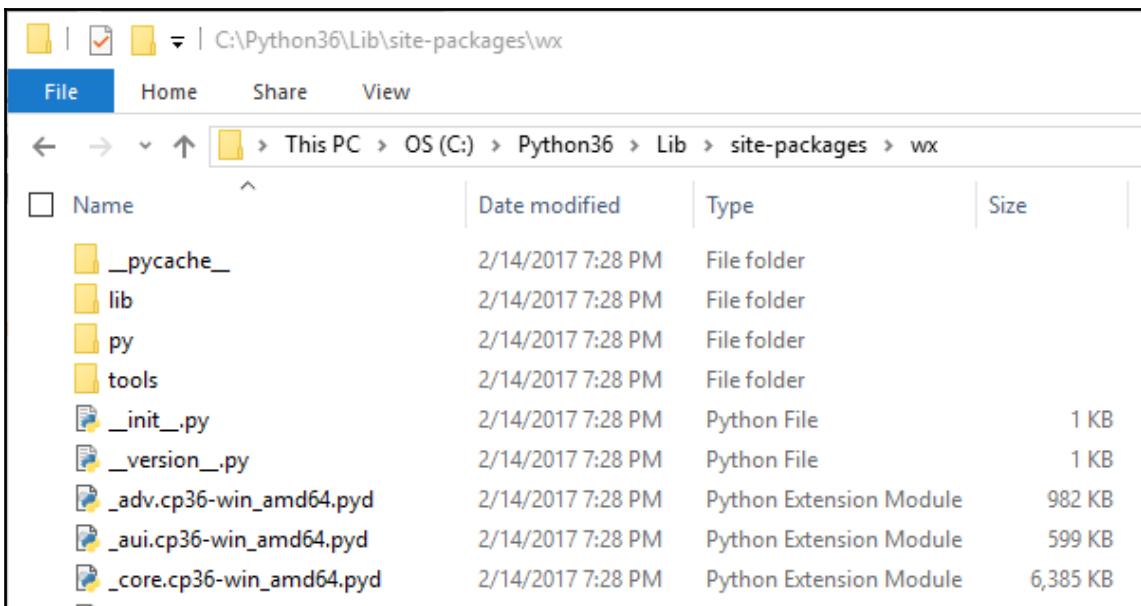
Per noi, la parte più importante di questo schema è che stiamo installando la build wxPython/Phoenix che è per Python 3.6 (il *cp36* nel nome del programma di installazione) e per il sistema operativo Windows a 64 bit (il *win_amd64* parte del nome dell'installatore).

Dopo aver scaricato con successo il pacchetto wxPython/Phoenix, possiamo ora navigare nella directory in cui risiede e installare questo pacchetto utilizzando pip:



```
C:\>pip install "wxPython_Phoenix-3.0.3.dev2808+717fdf9-cp36-cp36m-win_amd64.whl"
Processing c:\wxpython_phoenix-3.0.3.dev2808+717fdf9-cp36-cp36m-win_amd64.whl
Requirement already satisfied: six in c:\python36\lib\site-packages (from wxPython-Phoenix==3.0.3.dev2808+717fdf9)
Installing collected packages: wxPython-Phoenix
Successfully installed wxPython-Phoenix-3.0.3.dev2808+717fdf9
C:\>
```

Abbiamo una nuova cartella denominata wx nel nostro Python pacchetti-sito cartella:



wx è il nome della cartella in cui è stata installata la libreria wxPython Phoenix.
Importeremo questo modulo nel nostro codice Python.

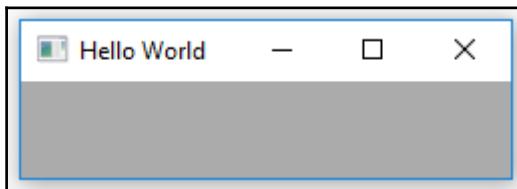
Possiamo verificare che la nostra installazione abbia funzionato eseguendo questo semplice script demo dal sito web ufficiale di wxPython/Phoenix. Il link al sito ufficiale è <http://wxpython.org/Phoenix/docs/html/>.

Considera il seguente codice:

```
importa wx
app = wx.App()
frame = wx.Frame(Nessuno, -1, "Hello World")
frame.Show()
app.MainLoop()
```

L'esecuzione dello script Python 3.6 precedente crea la seguente GUI utilizzando wxPython/Phoenix:

Hello_wxPython.py



Come funziona...

In questa ricetta, abbiamo installato con successo la versione corretta del toolkit wxPython che possiamo usare con Python 3.6. Abbiamo trovato il progetto Phoenix per questo toolkit GUI, che è la linea di sviluppo attuale e attiva. Phoenix sostituirà il classico toolkit wxPython nel tempo ed è particolarmente mirato a lavorare bene con Python 3.6.

Dopo aver installato con successo il toolkit wxPython/Phoenix, abbiamo quindi creato una GUI utilizzando questo toolkit in sole cinque righe di codice.

In precedenza abbiamo ottenuto gli stessi risultati utilizzando tkinter.



Creazione della nostra GUI in wxPython

In questa ricetta, inizieremo a creare le nostre GUI Python utilizzando il toolkit GUI wxPython.

Per prima cosa ricreeremo molti dei widget che abbiamo creato in precedenza utilizzando tkinter, fornito con Python.

Quindi, esploreremo alcuni dei widget offerti dal toolkit della GUI wxPython, che non sono così facili da creare usando tkinter.

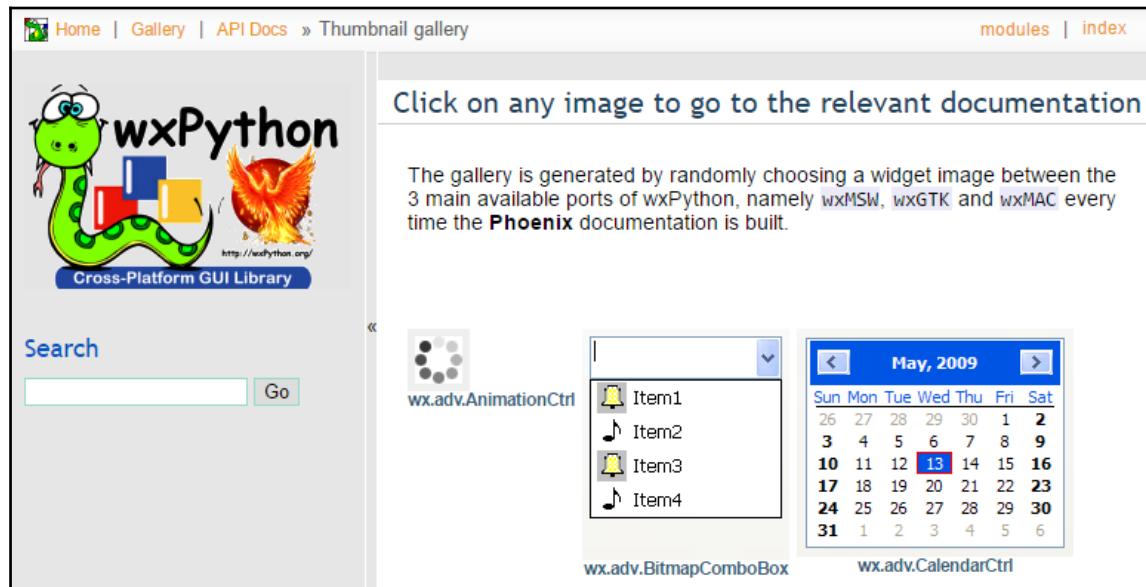
Prepararsi

La ricetta precedente ti ha mostrato come installare la versione corretta di wxPython che corrisponda sia alla tua versione di Python che al sistema operativo in esecuzione.

Come farlo...

Un buon punto di partenza per esplorare il toolkit della GUI wxPython è andare al seguente URL:
<http://wxpython.org/Phoenix/docs/html/gallery.html>

Questa pagina web mostra molti widget wxPython e, facendo clic su uno di essi, siamo portati alla loro documentazione, che è una funzionalità molto utile per conoscere rapidamente un controllo wxPython:



La seguente schermata mostra la documentazione per un widget pulsante wxPython:

The screenshot shows a web browser displaying the wxPython API documentation. The left sidebar features the wxPython logo and a 'Table Of Contents' section with links to various wx.Button-related pages. The main content area is titled 'Class API' and shows the definition of the `wx.Button` class, which inherits from `AnyButton`. It lists two possible constructors: `Button()` and `Button(parent, id=ID_ANY, label="", pos=DefaultPosition, size=DefaultSize, style=0, validator=DefaultValidator, name=ButtonNameStr)`. A descriptive text below explains that a button is a control containing a text string and one of the most common GUI elements.

Possiamo creare molto rapidamente una finestra di lavoro che viene fornita con un titolo, una barra dei menu e anche una barra di stato. Questa barra di stato mostra il testo di una voce di menu quando ci si passa sopra con il mouse. Ciò può essere ottenuto scrivendo il seguente codice:

```
# Importa toolkit GUI wxPython
import wx

# Sottoclasse frame wxPython
GUI di classe (wx.Frame):
    def __init__(self, parent, title, size=(200,100)):
        # Inizializza la superclasse
        wx.Frame.__init__(self, parent, title=title, size=size)

        # Cambia il colore di sfondo della cornice
        self.SetBackgroundColor('bianco')

        # Crea barra di stato
        self.CreateStatusBar()

        # Crea il menu
        menu= wx.Menu()

        # Aggiungi voci di menu al menu
        menu.Append(wx.ID_ABOUT, "About", "wxPython GUI")
        menu.AppendSeparator()
```

```
menu.Append(wx.ID_EXIT,"Esci"," Esci dalla GUI")

# Crea la barra dei menu
menuBar = wx.MenuBar()

# Dai un titolo al menu
menuBar.Append(menu,"File")

# Collega la barra dei menu alla cornice
self.SetMenuBar(menuBar)

# Visualizza la cornice
self.Mostra()

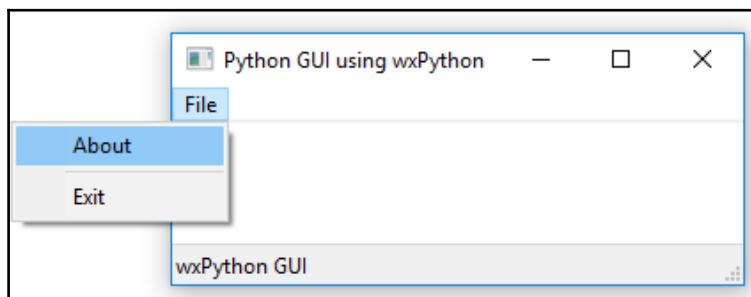
# Crea un'istanza dell'applicazione wxPython
app = wx.App()

# Chiama la GUI wxPython sottoclasse aumentando la dimensione predefinita della finestra
GUI (Nessuno, "GUI Python che utilizza wxPython", (300,150))

# Esegui il ciclo di eventi della GUI principale
app.MainLoop()
```

Questo crea la seguente GUI, che è scritta in Python usando il wxPython biblioteca:

GUI_wxPython.py



Nel codice precedente, abbiamo ereditato da `wx.Frame`. Nel prossimo codice, ereditiamo da `wx.Pannello` e passiamo dentro `wx.Frame` al `__dentro__()` metodo della nostra classe.



Nel wxPython, la finestra della GUI di primo livello è chiamata frame. Non può esserci una GUI wxPython senza un frame e il frame deve essere creato come parte di un'applicazione wxPython. Creiamo sia l'applicazione che il frame in fondo al nostro codice.

Per aggiungere widget alla nostra GUI, dobbiamo collegarli a un pannello. Il genitore del pannello è il frame (la nostra finestra di primo livello) e il genitore dei widget che mettiamo nel pannello è il pannello.

Il codice seguente aggiunge una multilinea casella di testo widget a un pannello il cui genitore è un frame. Aggiungiamo anche un widget pulsante al widget del pannello, che, se cliccato, stampa del testo al casella di testo.

Ecco il codice completo:

```
importa wx # Importa toolkit GUI wxPython
GUI di classe (wx.Panel): # Sottoclasse pannello wxPython
def __init__(self, genitore):
    # Inizializza la superclasse
    wx.Panel.__init__(self, genitore)

    # Crea barra di stato
    parent.CreateStatusBar()

    # Crea il menu
    menu= wx.Menu()

    # Aggiungi voci di menu al menu
    menu.Append(wx.ID_ABOUT, "About", "wxPython GUI")
    menu.AppendSeparator()
    menu.Append(wx.ID_EXIT,"Esci", " Esci dalla GUI")

    # Crea la barra dei menu
    menuBar = wx.MenuBar()

    # Dai un titolo al menu
    menuBar.Append(menu,"File")

    # Collega la barra dei menu alla cornice
    genitore.SetMenuBar(menuBar)

    # Crea un pulsante di stampa
    button = wx.Button(self, label="Print", pos=(0,60))

    # Pulsante Connetti per fare clic sul metodo Evento
    self.Bind(wx.EVT_BUTTON, self.printButton, pulsante)

    # Crea un widget di controllo del testo
    self.textBox = wx.TextCtrl(self, size=(280,50),
                               style=wx.TE_MULTILINE)
```

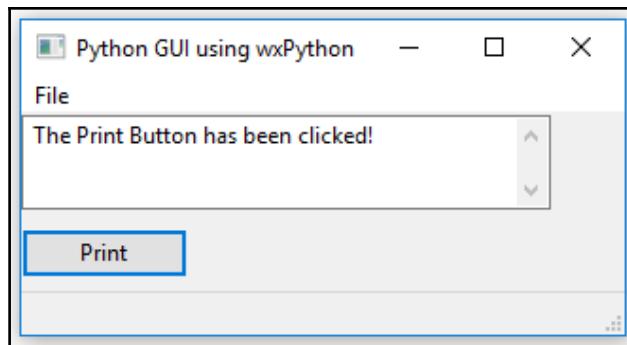
```
# gestore di eventi di callback
def printButton(self, evento):
    self.textBox.AppendText("Il pulsante Stampa è stato cliccato!")

app = wx.App()           # Crea un'istanza dell'applicazione wxPython
                         # Crea cornice
frame = wx.Frame(None, , size=(300,180)) GUI(frame) #
Passa il frame nella GUI
frame.Show() # Mostra il frame
app.MainLoop() # Esegue il ciclo di eventi della GUI principale
```

Nel codice precedente, genitore è un `wx.Frame` stiamo passando all'inizializzatore della GUI.

Eseguendo il codice precedente e facendo clic sul widget del pulsante wxPython si ottiene il seguente output della GUI:

`wxPython_panel_GUI.py`



Come funziona...

Abbiamo creato la nostra GUI in questa ricetta utilizzando il toolkit della GUI wxPython maturo. Con solo poche righe di codice Python, siamo stati in grado di creare una GUI completamente funzionale fornita con Riduci al minimo, massimizza, e Uscita pulsanti. Abbiamo aggiunto una barra dei menu, un controllo di testo multilinea e un pulsante. Abbiamo anche creato una barra di stato che mostra il testo quando selezioniamo una voce di menu. Abbiamo inserito tutti questi widget in un widget contenitore del pannello.

Abbiamo collegato il pulsante per stampare al controllo di testo.

Quando si passa con il mouse su una voce di menu, nella barra di stato viene visualizzato del testo.

Aggiunta rapida di controlli utilizzando wxPython

In questa ricetta, ricreeremo la GUI che abbiamo originariamente creato in precedenza in questo libro con tkinter, ma questa volta utilizzeremo la libreria wxPython. Vedremo quanto è facile e veloce usare il toolkit della GUI wxPython per creare le nostre GUI Python.

Non ricreeremo l'intera funzionalità che abbiamo creato nei capitoli precedenti. Ad esempio, non internazionalizzeremo la nostra GUI wxPython, né la collegheremo a un database MySQL. Ricreeremo gli aspetti visivi della GUI e aggiungeremo alcune funzionalità.



Il confronto di diverse librerie ci dà la possibilità di scegliere quali toolkit utilizzare per lo sviluppo della nostra GUI Python e possiamo combinare molti di questi toolkit nel nostro codice Python.

Prepararsi

Assicurati di avere installato il modulo wxPython per seguire questa ricetta.

Come farlo...

Per prima cosa, creiamo il nostro Python OOP class come abbiamo fatto prima, usando tkinter, ma questa volta ereditiamo ed estendiamo da wx.Frame classe. Per ragioni di chiarezza, non chiamiamo più la nostra classe OOP ma, invece, la rinominiamo comeMainframe.



Nel wxPython, la finestra principale della GUI è chiamata Frame.

Creiamo anche un metodo di callback che chiude la GUI quando facciamo clic su **Uscita** voce di menu e dichiarare un grigio chiaro tupla come colore di sfondo per la nostra GUI:

```
importa wx  
COLORE DI SFONDO = (240, 240, 240, 255)  
  
classe MainFrame(wx.Frame):  
    def __init__(self, *args, **kwargs):  
        wx.Frame.__init__(self, *args, **kwargs)  
        self.createWidgets()  
        self.Mostra()
```

```
def exitGUI(self, evento):           # richiama
    auto.Distruzione()

def createWidgets(self):
    self.CreateStatusBar()          # Metodo integrato wxPython
    self.createMenu()
    self.createNotebook()
```

Quindi aggiungiamo un controllo a schede alla nostra GUI creando un'istanza di wxPython Taccuino class e assegnalo come genitore della nostra classe personalizzata denominata Widget.

Il taccuino la variabile di istanza della classe ha wx.Pannello come suo genitore:

```
# -----
def createNotebook(self):
    pannello = wx.Panel(self)
    notebook = wx.Notebook(pannello)
    widgets = Widgets(notebook) # Classe personalizzata spiegata sotto
    notebook.AddPage(widgets, "Widgets")
    notebook.SetBackgroundColour(BACKGROUNDCOLOR)
    # disposizione
    boxSizer = wx.BoxSizer() boxSizer.Add(notebook,
    1, wx.EXPAND) panel.SetSizerAndFit(boxSizer)
```

In wxPython, il widget a schede è chiamato Taccuino, proprio come in tkinter.



Ogni Taccuino il widget deve avere un genitore e, per disporre i widget nel Taccuino in wxPython, usiamo diversi tipi di misuratori.

wxPython misuratori sono gestori di layout, simili al gestore di layout della griglia di tkinter.



Successivamente, aggiungiamo i controlli alla nostra pagina Notebook e lo facciamo creando una classe separata che eredita da wx.Pannello:

Widget di classe (wx.Panel):

```
def __init__(self, genitore):
    wx.Panel.__init__(self, parent)
    self.createWidgetsFrame()
    self.addWidgetes()
    self.layoutWidgets()
```

Modifichiamo il nostro codice GUI suddividendolo in piccoli metodi, seguendo il protocollo di programmazione Python OOP, che mantiene il nostro codice gestibile e comprensibile:

```
# -----
def createWidgetsFrame(self):
    self.pannello = wx.Panel(self)
    staticBox = wx.StaticBox(self.panel, -1, "Widgets Frame") self.statBoxSizerV =
        wx.StaticBoxSizer(staticBox, wx.VERTICAL)
# -----
def layoutWidgets(self):
    boxSizerV = wx.BoxSizer(wx.VERTICAL)
    boxSizerV.Add(self.statBoxSizerV, 1, wx.ALL)
    self.panel.SetSizer(boxSizerV)
    boxSizerV.SetSizeHints(self.panel)
# -----
def addWidgets(self):
    self.addCheckBox()
    self.addRadioButtons()
    self.addStaticBoxWithLabels()
```



Quando si utilizza wxPython StaticBox widget, per disporli con successo, usiamo una combinazione di StaticBoxSizer e un normale BoxSizer. Il wxPython StaticBox è molto simile al tkinter EtichettaFrame aggeggio.

Incorporamento StaticBox dentro un altro StaticBox è semplice in tkinter, ma usare wxPython è un po' non intuitivo. Un modo per farlo funzionare è mostrato nel seguente frammento di codice:

```
def addStaticBoxWithLabels(self):
    boxSizerH = wx.BoxSizer(wx.HORIZONTAL)
    staticBox = wx.StaticBox(self.panel, -1, "Etichette all'interno di un frame")
    staticBoxSizerV = wx.StaticBoxSizer(staticBox, wx.VERTICAL) boxSizerV =
        wx.BoxSizer( wx.VERTICAL )
    staticText1 = wx.StaticText(self.panel, -1, "Scegli un numero:")
    boxSizerV.Add(staticText1, 0, wx.ALL)
    staticText2 = wx.StaticText(self.panel, -1,"Label 2")
    boxSizerV.Add(staticText2, 0, wx.ALL)
    # -----
    staticBoxSizerV.Add(boxSizerV, 0, wx.ALL)
    boxSizerH.Add(staticBoxSizerV)
    # -
    boxSizerH.Add(wx.TextCtrl(self.panel))
    # Aggiungi boxSizer locale al frame principale
    self.statBoxSizerV.Add(boxSizerH, 1, wx.ALL)
```

Per prima cosa, creiamo un orizzontale BoxSizer. Successivamente, creiamo una verticale StaticBoxSizer perché vogliamo disporre due etichette in un layout verticale in questa cornice.

Per disporre un altro widget a destra dell'incorporato StaticBox, dobbiamo assegnare entrambi gli embedded StaticBox con i suoi controlli figli e il widget successivo all'orizzontale Formato scatola e poi assegna questo BoxSizer, che ora contiene sia il nostro embedded StaticBox e gli altri nostri widget, al principale StaticBox.

Questo suona confuso?

Devi solo sperimentare con questi misuratori per avere un'idea di come usarli. Inizia con il codice per questa ricetta e commenta un po' di codice o modificalo qualcuno *Xe si* coordinate per vedere gli effetti.

È anche bene leggere la documentazione ufficiale di wxPython per saperne di più.



L'importante è sapere dove aggiungere al diverso misuratori nel codice per ottenere il layout che desideriamo.

Per creare il secondo StaticBox sotto il primo, creiamo separato StaticBoxSizers e assegnarli allo stesso pannello:

```
Widget di classe (wx.Panel):
def __init__(self, genitore):
    wx.Panel.__init__(self, parent) self.panel =
        wx.Panel(self) self.createWidgetsFrame()
        self.createManageFilesFrame()
        self.addWidgets()

    self.addFileWidgets()
    self.layoutWidgets()
# -----
def createWidgetsFrame(self):
    staticBox = wx.StaticBox(self.panel, -1, "Widget Frame",
                           dimensione=(285, -1))
    self.statBoxSizerV = wx.StaticBoxSizer(staticBox, wx.VERTICAL)
# -----
def createManageFilesFrame(self):
    staticBox = wx.StaticBox(self.panel, -1, "Gestisci file",
                           dimensione=(285, -1))
    self.statBoxSizerMgrV = wx.StaticBoxSizer(staticBox, wx.VERTICAL)
# -----
def layoutWidgets(self):
    boxSizerV = wx.BoxSizer( wx.VERTICAL )
```

```
boxSizerV.Add( self.statBoxSizerV, 1, wx.ALL )
boxSizerV.Add( self.statBoxSizerMgrV, 1, wx.ALL )

self.panel.SetSizer( boxSizerV )
boxSizerV.SetSizeHints( self.panel )

# -----
def addFileWidgets(self):
    boxSizerH = wx.BoxSizer(wx.HORIZONTAL)
    boxSizerH.Add(wx.Button(self.panel, label='Browse to File...'))
    boxSizerH.Add(wx.TextCtrl(self.panel, size=(174, -1),
                             valore= "Z:"))

    boxSizerH1 = wx.BoxSizer(wx.HORIZONTAL)
    boxSizerH1.Add(wx.Button(self.panel, label='Copia file in: '))
    boxSizerH1.Add(wx.TextCtrl(self.panel, size=(174, -1),
                             valore="Z:Backup"))

    boxSizerV = wx.BoxSizer(wx.VERTICAL)
    boxSizerV.Add(boxSizerH)
    boxSizerV.Add(boxSizerH1)

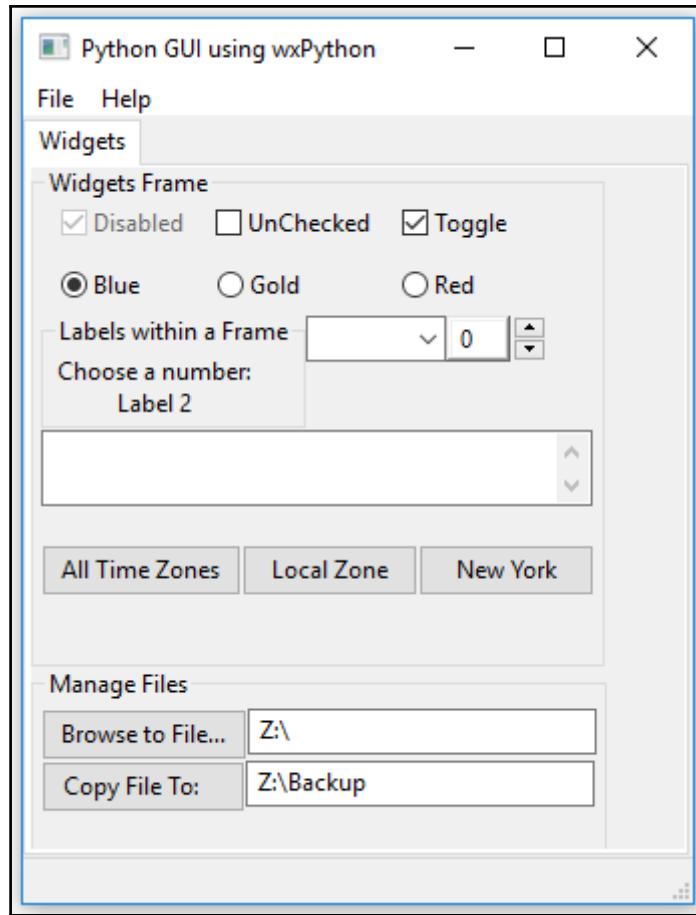
    self.statBoxSizerMgrV.Add( boxSizerV, 1, wx.ALL )
```

Il codice seguente crea un'istanza del ciclo di eventi principale, che esegue il nostro programma GUI wxPython:

```
# =====
# Avvia GUI
# =====
app = wx.App()
MainFrame(Nessuno, , size=(350,450))
app.MainLoop()
```

Il risultato finale della nostra GUI wxPython è il seguente:

GUI_wxPython.py



Come funziona...

Progettiamo e impostiamo la nostra GUI wxPython in diverse classi.

Fatto ciò, nella sezione inferiore del nostro modulo Python, creiamo un'istanza dell'applicazione wxPython. Successivamente, istanziamo il nostro codice GUI wxPython.

Successivamente, chiamiamo il ciclo di eventi della GUI principale, che esegue tutto il nostro codice Python in esecuzione all'interno di questo processo dell'applicazione. Questo mostra la nostra GUI di wxPython.



Qualunque sia il codice che inseriamo tra la creazione dell'app e la chiamata al suo ciclo di eventi principale, diventa la nostra GUI wxPython. Potrebbe volerci del tempo per abituarsi davvero alla libreria wxPython e alla sua API, ma una volta capito come usarla, questa libreria è davvero divertente e uno strumento potente per costruire le nostre GUI Python. C'è anche uno strumento di progettazione visiva che può essere utilizzato con wxPython:
<http://www.cae.tntech.edu/help/programming/wxdesigner-getting-started/view>

Questa ricetta ha usato l'OOP per imparare a usare il toolkit della GUI di wxPython.

Cercando di incorporare un'app wxPython principale in un'app tkinter principale

Ora che abbiamo creato la stessa GUI utilizzando sia la libreria tkinter incorporata di Python che il wrapper wxPython del wxWidget libreria, abbiamo davvero bisogno di combinare le GUI che abbiamo creato utilizzando queste tecnologie.



Sia le librerie wxPython che tkinter hanno i loro vantaggi. Nei forum online come <http://stackoverflow.com/>, spesso vediamo domande, come quale sia la migliore, quale toolkit GUI dovrei usare e così via. Questo suggerisce che dobbiamo prendere una decisione aut-aut. Non dobbiamo prendere una decisione del genere.

Una delle sfide principali nel fare ciò è che ogni toolkit della GUI deve avere il proprio ciclo di eventi.

In questa ricetta, proveremo a incorporare una semplice GUI wxPython chiamandola dalla nostra GUI tkinter.

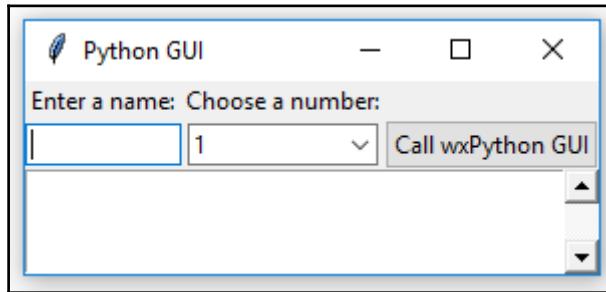
Prepararsi

Riutilizzeremo la GUI di tkinter che abbiamo creato nella ricetta *Widget della casella combinata*, nel Capitolo 1, *Creazione del modulo GUI e aggiunta di widget*.

Come farlo...

Inizieremo da una semplice GUI di tkinter, che ha il seguente aspetto:

Incorpora_wxPython.py



Successivamente, proveremo a invocare una semplice GUI wxPython, che abbiamo creato in una ricetta precedente in questo capitolo.

Quello che segue è l'intero codice per farlo in un modo semplice e non OOP:

```
# =====
# importa tkinter come tk
# da tkinter import ttk, scrolledtext

vittoria = tk.Tk()
win.title ("GUI Python")
aLabel = ttk.Label(win, text="A Label")
aLabel.grid(column=0, row=0)
ttk.Label(win, text="Inserisci un nome:").grid(column=0, row=0) name =
tk.StringVar()
nameEntered = ttk.Entry(win, width=12, textvariable=name)
nameEntered.grid(column=0, row=1)
ttk.Label(win, text="Scegli un numero:").grid(column=1, row=0) numero =
tk.StringVar()
numberChosen = ttk.Combobox(win, width=12, textvariable=numero)
numberChosen['values'] = (1, 2, 4, 42, 100)
numeroScelto.grid(colonna=1, riga=1)
numeroScelto.corrente(0)
scrollW = 30
scrollH = 3
scr = scrolledtext.ScrolledText(win, width=scrollW, height=scrollH, wrap=tk.WORD)

scr.grid(column=0, sticky='WE', columnspan=3)
nameEntered.focus()
```

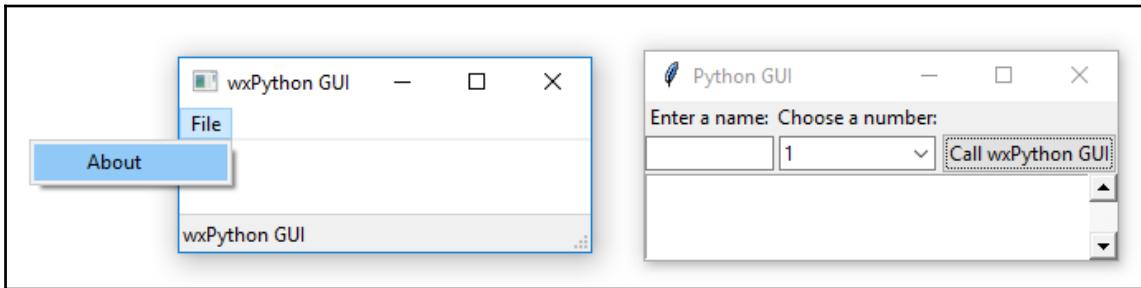
```
# ===== def
wxPythonApp():
    importa wx
    app = wx.App()
    frame = wx.Frame(None, -1, "wxPython GUI", size=(200,150))
    frame.SetBackgroundColour('bianco')
    frame.CreateStatusBar()
    menu= wx.Menu()
    menu.Append(wx.ID_ABOUT, "About", "wxPython GUI") menuBar
    = wx.MenuBar()
    menuBar.Append(menu,"File")
    frame.SetMenuBar(menuBar)
    frame.Show()
    app.MainLoop()

action = ttk.Button (win, text = "Call wxPython GUI", command = wxPythonApp) action.grid
(colonna = 2, riga = 1)

# =====
# Avvia GUI
# =====
win.mainloop()
```

L'esecuzione del codice precedente avvia una GUI wxPython dalla nostra GUI tkinter dopo aver fatto clic sul pulsante di controllo tkinter:

Incorpora_wxPython.py



Come funziona...

La parte importante è che abbiamo inserito l'intero codice wxPython nella sua funzione, che abbiamo chiamato `def wxPythonApp()`.

Nella funzione di callback per l'evento click del pulsante, chiamiamo semplicemente questo codice.



Una cosa da notare è che dobbiamo chiudere la GUI di wxPython prima di poter continuare a usare la GUI di tkinter.

Cercando di incorporare il nostro codice GUI di tkinter in wxPython

In questa ricetta, andremo nella direzione opposta alla ricetta precedente e proveremo a chiamare il nostro codice della GUI di tkinter dall'interno di una GUI di wxPython.

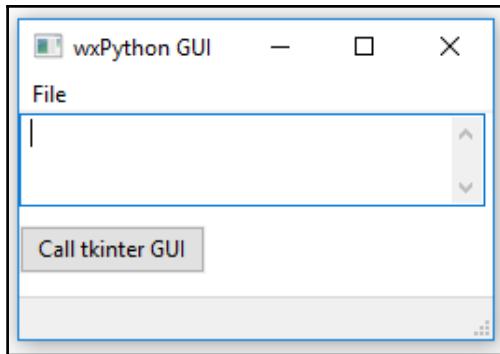
Prepararsi

Riutilizzeremo parte del codice della GUI di wxPython che abbiamo creato in una ricetta precedente in questo capitolo.

Come farlo...

Inizieremo da una semplice GUI wxPython, che ha il seguente aspetto:

Incorpora_tkinter.py



Successivamente, proveremo a invocare una semplice GUI di tkinter.

Quello che segue è l'intero codice per farlo in un modo semplice e non OOP:

```
# ===== def
tkinterApp():
    import tkinter as tk
    from tkinter import ttk
    win = tk.Tk()

    win.title ("GUI Python")
    aLabel = ttk.Label(win, text="A Label")
    aLabel.grid(column=0, row=0)
    ttk.Label(win, text="Inserisci un nome:").grid(column=0, row=0) name =
    tk.StringVar()
    nameEntered = ttk.Entry(win, width=12, textvariable=name)
    nameEntered.grid(column=0, row=1)
    nomeEntrato.focus()

    def buttonCallback():
        action.configure(text='Hello ' + name.get())
        action = ttk.Button(win, text="Print", command=buttonCallback)
        action.grid(column=2, riga=1)
        win.mainloop()

# =====
importa wx
app = wx.App()
frame = wx.Frame(None, -1, "wxPython GUI", size=(270,180))
frame.SetBackgroundColour('bianco')
frame.CreateStatusBar()
menu= wx.Menu()
menu.Append(wx.ID_ABOUT, "About", "wxPython GUI") menuBar
= wx.MenuBar()
menuBar.Append(menu,"File")
frame.SetMenuBar(menuBar)
textBox = wx.TextCtrl(frame, size=(250,50), style=wx.TE_MULTILINE)

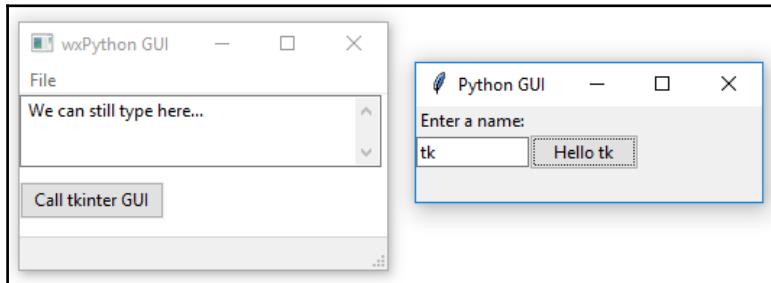
def tkinterEmbed(evento):
    tkinterApp()

    button = wx.Button(frame, label="Call tkinter GUI", pos=(0,60))
    frame.Bind(wx.EVT_BUTTON, tkinterEmbed, button)
    cornice.Mostra()

# =====
# Avvia la GUI di wxPython
# =====
app.MainLoop()
```

L'esecuzione del codice precedente avvia una GUI tkinter dalla nostra GUI wxPython dopo aver fatto clic sul widget del pulsante wxPython. Possiamo quindi inserire del testo nella casella di testo della GUI di tkinter e, facendo clic sul suo pulsante, il testo del pulsante viene aggiornato con il nome:

Incorpora_tkinter.py



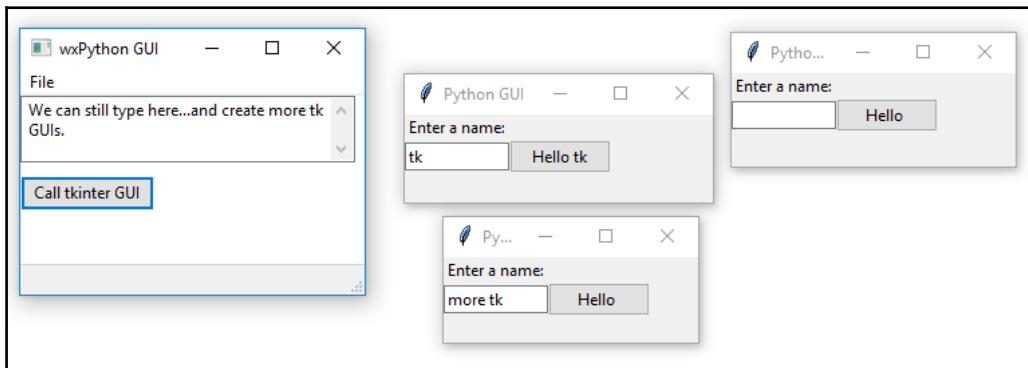
Dopo aver avviato il ciclo di eventi tkinter, la GUI di wxPython è ancora reattiva perché possiamo digitare nel TextCtrl widget mentre la GUI di tkinter è attiva e funzionante.



Nella ricetta precedente, non potevamo usare la nostra GUI di tkinter finché non avessimo chiuso la GUI di wxPython. Essere consapevoli di questa differenza può aiutare le nostre decisioni di progettazione se vogliamo combinare le due tecnologie GUI Python.

Possiamo anche creare diverse istanze della GUI di tkinter facendo clic più volte sul pulsante della GUI di wxPython. Non possiamo, tuttavia, chiudere la GUI di wxPython mentre sono ancora in esecuzione le GUI di tkinter. Dobbiamo chiuderli prima:

Incorpora_tkinter.py



Come funziona...

In questa ricetta, siamo andati nella direzione opposta alla ricetta precedente creando prima una GUI usando wxPython e poi, dall'interno, creando diverse istanze GUI costruite usando tkinter.

La GUI di wxPython è rimasta reattiva mentre uno o più tkinter Le GUI erano in esecuzione. Tuttavia, facendo clic sul pulsante tkinter si aggiornava solo il testo del pulsante in prima istanza.

Utilizzo di Python per controllare due diversi framework GUI GUI

In questa ricetta, esploreremo i modi per controllare i framework GUI tkinter e wxPython da Python. Abbiamo già utilizzato il modulo di threading Python per mantenere la nostra GUI reattiva in un capitolo precedente, *Thread e reti*, quindi qui cercheremo di usare lo stesso approccio.

Vedremo che le cose non funzionano sempre in un modo che sarebbe intuitivo.

Tuttavia, miglioreremo la nostra GUI di tkinter in modo che non risponda mentre invochiamo un'istanza della GUI di wxPython dall'interno di essa.

Prepararsi

Questa ricetta estenderà una ricetta precedente di questo capitolo, *Cercando di incorporare un'app wxPython principale in un'app tkinter principale*, in cui abbiamo provato a incorporare una GUI principale di wxPython nella nostra GUI di tkinter.

Come farlo...

Quando abbiamo creato un'istanza di una GUI wxPython dalla nostra GUI di tkinter, non potevamo più utilizzare i controlli della GUI di tkinter finché non avessimo chiuso l'unica istanza della GUI di wxPython. Miglioriamo su questo ora.

Il nostro primo tentativo potrebbe essere quello di utilizzare il threading dalla funzione di callback del pulsante tkinter.

Ad esempio, il nostro codice potrebbe avere il seguente aspetto:

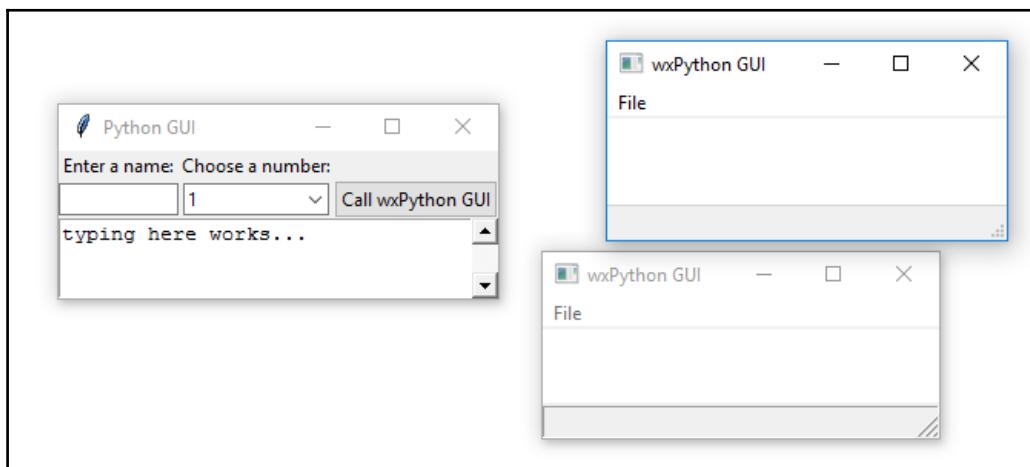
```
def wxPythonApp():
    importa wx
    app = wx.App()
    frame = wx.Frame(None, -1, "wxPython GUI", size=(200,150))
    frame.SetBackgroundColour('bianco')
    frame.CreateStatusBar()
    menu= wx.Menu()
    menu.Append(wx.ID_ABOUT, "About", "wxPython GUI") menuBar
    = wx.MenuBar()
    menuBar.Append(menu,"File")
    frame.SetMenuBar(menuBar)
    frame.Show()
    app.MainLoop()

def tryRunInThread():
    runT = Thread(target=wxPythonApp)
    runT.setDaemon(True)
    runT.start()
    print(runT)
    print('createThread():', runT.isAlive())

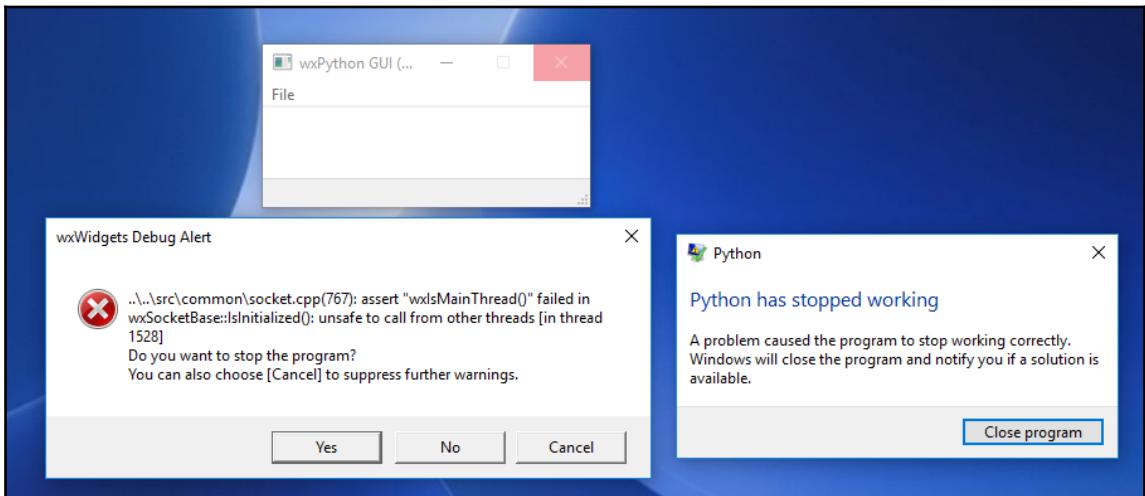
action = ttk.Button(win, text="Call wxPython GUI", command=tryRunInThread)
```

All'inizio, questo sembra funzionare, il che sarebbe intuitivo, poiché i controlli di tkinter non sono più disabilitati e possiamo creare diverse istanze della GUI di wxPython facendo clic sul pulsante. Possiamo anche digitare nella GUI di wxPython e selezionare gli altri widget di tkinter:

Control_Frameworks_NOT_working.py



Tuttavia, una volta che proviamo a chiudere le GUI, riceviamo un errore da wxWidget, e il nostro eseguibile Python si blocca:



Per evitare ciò, invece di provare a eseguire l'intera applicazione wxPython in un thread, possiamo modificare il codice per rendere solo il App wxPython.MainLoop eseguire in un thread:

```
def wxPythonApp():
    importa wx
    app = wx.App()
    frame = wx.Frame(None, -1, "wxPython GUI", size=(200,150))
    frame.SetBackgroundColour('bianco')
    frame.CreateStatusBar()
    menu= wx.Menu()
    menu.Append(wx.ID_ABOUT, "About", "wxPython GUI") menuBar
    = wx.MenuBar()
    menuBar.Append(menu,"File")
    frame.SetMenuBar(menuBar)
    frame.Show()

    runT = Thread(target=app.MainLoop)
    runT.setDaemon(True)
    runT.start()
    print(runT)
    print('createThread():', runT.isAlive())

action = ttk.Button (win, text = "Call wxPython GUI", command = wxPythonApp) action.grid
(colonna = 2, riga = 1)
```

Come funziona...

Per prima cosa abbiamo provato a eseguire l'intera applicazione GUI wxPython in un thread, ma non ha funzionato perché il ciclo di eventi principali di wxPython si aspetta di essere il thread principale dell'applicazione.

Abbiamo trovato una soluzione per questo eseguendo solo wxPython app.MainLoop in un thread che lo induce a credere che sia il thread principale.

Un effetto collaterale di questo approccio è che non possiamo più chiudere singolarmente tutte le istanze della GUI di wxPython. Almeno uno di essi si chiude solo quando chiudiamo la GUI di wxPython che ha creato i thread come demoni. Puoi provarlo facendo clic su**Chiama la GUI di wxPython** una o più volte e quindi provare a chiudere tutti i moduli finestra wxPython creati. Non possiamo chiudere l'ultimo finché non chiudiamo la GUI di tkinter chiamante!

Non sono abbastanza sicuro del motivo per cui questo è. Intuitivamente, ci si potrebbe aspettare di essere in grado di chiudere tutti i thread del demone senza dover attendere che il thread principale che li ha creati si chiuda per primo.

Probabilmente ha a che fare con un contatore di riferimento che non è stato impostato su zero mentre il nostro thread principale è ancora in esecuzione.

A livello pragmatico, è così che funziona attualmente.

Comunicazione tra le due GUI collegate

Nelle ricette precedenti, abbiamo trovato modi per connettere una GUI wxPython con una GUI tkinter, invocando l'una dall'altra e viceversa.

Sebbene entrambe le GUI funzionassero correttamente contemporaneamente, non comunicavano realmente tra loro, poiché si stavano solo avviando l'una con l'altra.

In questa ricetta, esploreremo i modi per far dialogare le due GUI.

Prepararsi

La lettura di una delle ricette precedenti potrebbe essere una buona preparazione per questa ricetta.

In questa ricetta, utilizzeremo un codice GUI leggermente modificato rispetto alla ricetta precedente, ma la maggior parte del codice di base per la creazione della GUI è lo stesso.

Come farlo...

Nelle ricette precedenti, una delle nostre sfide principali era come combinare due tecnologie GUI progettate per essere l'unico toolkit GUI per un'applicazione. Abbiamo trovato vari modi semplici per combinarli.

Lanceremo nuovamente la GUI di wxPython da un ciclo di eventi principali della GUI di tkinter e avvieremo la GUI di wxPython nel proprio thread, che viene eseguito all'interno del processo Python.exe.

Per fare ciò, utilizzeremo una coda Python multiprocessore globale condivisa.



Anche se spesso è meglio evitare i dati globali, in questa ricetta è una soluzione pratica e i globali Python sono davvero globali solo nel modulo in cui sono stati dichiarati.

Ecco il codice Python che fa comunicare tra loro le due GUI in una certa misura. Per risparmiare spazio, questo non è puro codice OOP. Né stiamo mostrando il codice di creazione per tutti i widget. Quel codice è lo stesso delle ricette precedenti:

```
# Comunicare.py
import tkinter as tk from tkinter
import ttk from threading import
Thread

vittoria = tk.Tk()
win.title ("GUI Python")

from queue import Queue
sharedQueue = Queue()
dataInQueue = False

def putDataIntoQueue(data):
    dati globaliInQueue
    dataInQueue = True
    sharedQueue.put(dati)

def readDataFromQueue():
```

```
dati globaliInQueue
dataInQueue = False
return sharedQueue.get()
# =====
import wx
GUI di classe (wx.Panel):
    def __init__(self, genitore):
        wx.Panel.__init__(self, parent)
        parent.CreateStatusBar()
        button = wx.Button(self, label="Print", pos=(0,60)) self.Bind(wx.EVT_BUTTON,
        self.writeToSharedQueue, button)

    # -----
def writeToSharedQueue(self, event):
    self.textBox.AppendText("Il pulsante Print è stato cliccato!n") putDataIntoQueue('Hi
from wxPython via Shared Queue.n') if dataInQueue:

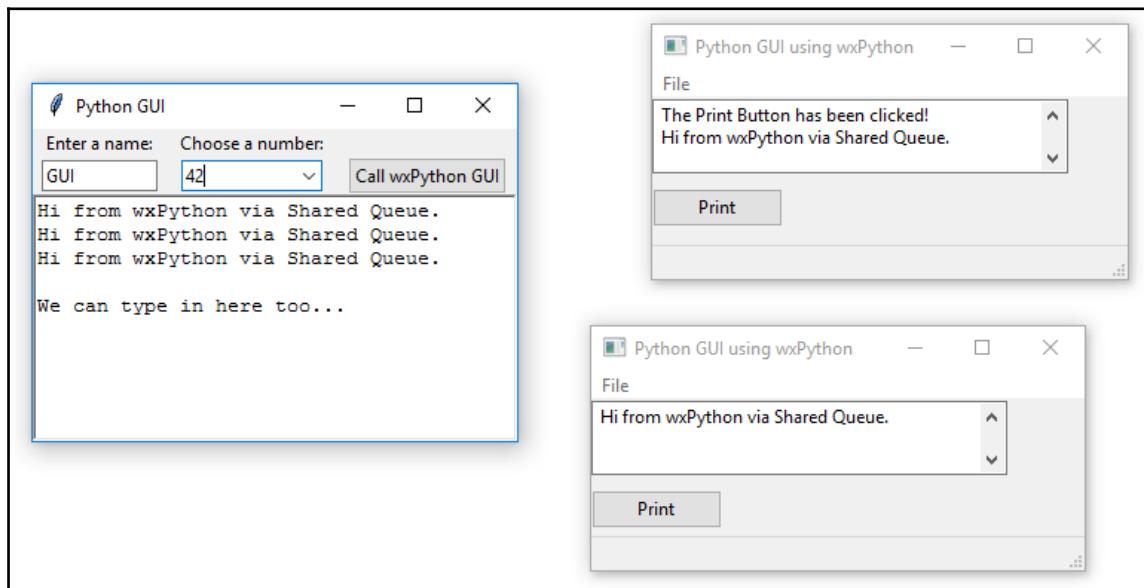
        data = readDataFromQueue()
        self.textBox.AppendText(data)
        text.insert('0.0', data) # inserisce i dati nella GUI di tkinter

# =====
def
wxPythonApp():
    app = wx.App()
    frame = wx.Frame(Nessuno, ,
                      dimensione=(300,180))
    GUI (cornice)
    cornice.Mostra()
    runT = Thread(target=app.MainLoop)
    runT.setDaemon(True)
    runT.start()
    print(runT)
    print('createThread():', runT.isAlive())
# =====
action = ttk.Button (win, text = "Call wxPython GUI", command = wxPythonApp) action.grid
(colonna = 2, riga = 1)

# =====
# Avvia GUI
# =====
win.mainloop()
```

L'esecuzione del codice precedente crea prima la parte tkinter del programma e, quando si fa clic sul pulsante in questa GUI, viene eseguita la GUI wxPython. Entrambi sono in esecuzione contemporaneamente a prima ma, questa volta, c'è un ulteriore livello di comunicazione tra le due GUI:

Comunicare.py



La GUI di tkinter è mostrata sul lato sinistro nello screenshot precedente e facendo clic su **Chiama la GUI di wxPython** pulsante, invochiamo un'istanza della GUI di wxPython. Possiamo creare più istanze facendo clic più volte sul pulsante.

Tutte le GUI create rimangono reattive. Non si bloccano né si bloccano.



Facendo clic su **Stampa** pulsante su una qualsiasi delle istanze della GUI di wxPython scrive una frase nella propria TextCtrl widget e poi scrive un'altra riga su se stesso, oltre che sulla GUI di tkinter. Dovrai scorrere verso l'alto per vedere la prima frase nella GUI di wxPython.



Il modo in cui funziona è utilizzando un livello di modulo coda e un tkinter Testo aggeggio.

Un elemento importante da notare è che creiamo un thread per eseguire wxPython app.MainLoop, come abbiamo fatto nella ricetta precedente:

```
def wxPythonApp():
    app = wx.App()
    frame = wx.Frame(Nessuno,
                      dimensione=(300,180))
    GUI (cornice)
    cornice.Mostra()
    runT = Thread(target=app.MainLoop)
    runT.setDaemon(True)
    runT.start()
```

Creiamo una classe che eredita da wx.Pannello e chiamalo GUI e quindi istanziare un'istanza di questa classe nel codice precedente.

Creiamo un metodo di callback dell'evento click-click in questa classe, che poi chiama il codice procedurale che è stato scritto sopra di esso. Per questo motivo, la classe ha accesso alle funzioni e può scrivere nella coda condivisa:

```
# -----
def writeToSharedQueue(self, event):
    self.textBox.AppendText("Il pulsante Print è stato cliccato!\n") putDataIntoQueue('Hi
from wxPython via Shared Queue.\n') if dataInQueue:

    data = readDataFromQueue()
    self.textBox.AppendText(data)
    text.insert('0.0', data) # inserisce i dati in tkinter
```

Per prima cosa controlliamo se i dati sono stati inseriti nella coda condivisa nel metodo precedente e, in tal caso, stampiamo i dati comuni su entrambe le GUI.



Il putDataIntoQueue() line inserisce i dati nella coda e readDataFromQueue() lo rilegge, salvandolo nel dati variabile. text.insert('0.0', dati) è la linea che scrive questi dati nella GUI di tkinter dal **Stampa** metodo di callback wxPython del pulsante.

Le seguenti sono le funzioni procedurali (non i metodi, perché non sono vincolate) che vengono chiamate nel codice e lo fanno funzionare:

```
from multiprocessing import Queue
sharedQueue = Queue()
dataInQueue = False

def putDataIntoQueue(data):
    dati globaliInQueue
    dataInQueue = True
```

```
sharedQueue.put(dati)

def readDataFromQueue():
    dati globaliInQueue
    dataInQueue = False
    return sharedQueue.get()
```

Abbiamo usato un semplice flag booleano chiamato `dataInQueue` per comunicare quando i dati sono disponibili in coda.

Come funziona...

In questa ricetta, abbiamo combinato con successo le due GUI che abbiamo creato in modo simile, ma che in precedenza erano indipendenti e non parlavano tra loro. Tuttavia, in questa ricetta, li abbiamo collegati ulteriormente facendo avviare un'altra GUI e, tramite un semplice meccanismo di coda Python multiprocessore, siamo stati in grado di farli comunicare tra loro, scrivendo dati da una coda condivisa in entrambe le GUI.

Sono disponibili molte tecnologie più avanzate e complicate per connettere diversi processi, thread, pool, lock, pipe, connessioni TCP/IP e così via.

Nello spirito Pythonic, abbiamo trovato una soluzione semplice che funziona per noi. Una volta che il nostro codice diventa più complicato, potremmo doverlo rifactorizzare, ma questo è un buon inizio.

10

Creazione di fantastiche GUI 3D con PyOpenGL e PyGlet

In questo capitolo creeremo fantastiche GUI Python che mostrano vere immagini tridimensionali che possono essere ruotate su se stesse in modo da poterle guardare da tutti i lati. Tratteremo le seguenti ricette:

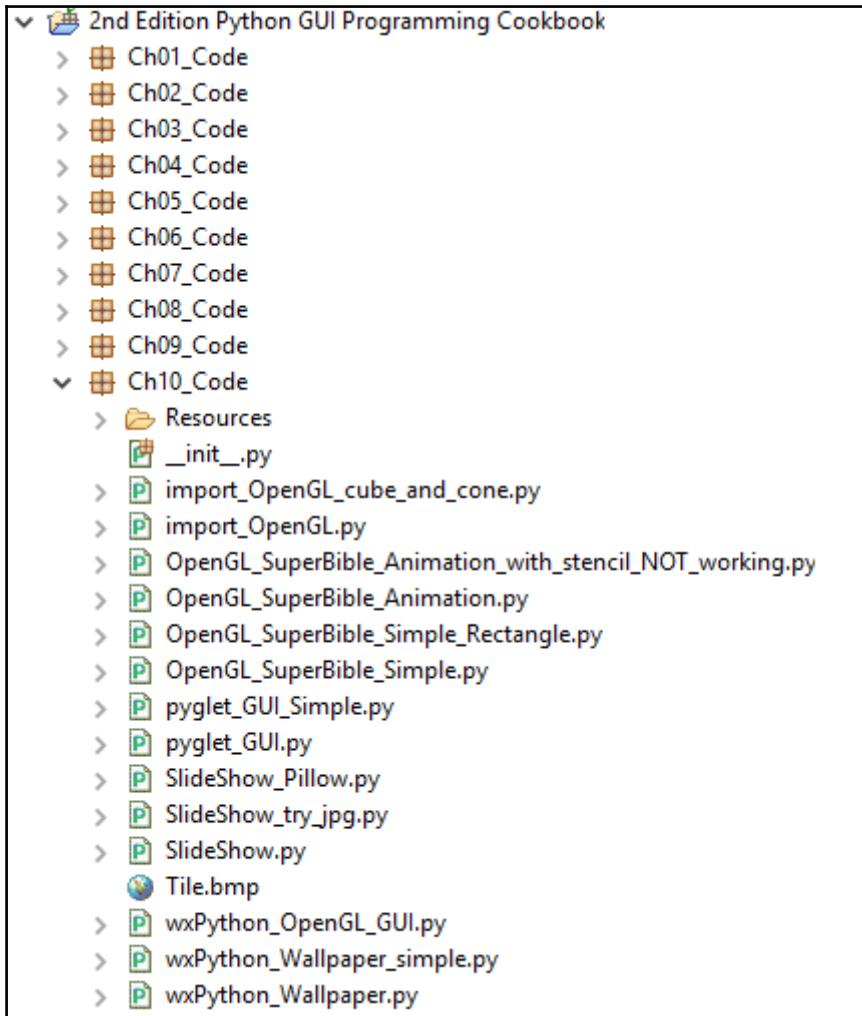
- PyOpenGL trasforma la nostra GUI La nostra GUI in 3D!
- Usare le bitmap per rendere carina la nostra GUI
- PyGlet trasforma la nostra GUI più facilmente di PyOpenGL La nostra GUI in colori sorprendenti
- Animazione OpenGL
- Creare una presentazione usando tkinter

introduzione

In questo capitolo, trasformeremo la nostra GUI conferendole vere capacità tridimensionali. Useremo due pacchetti Python di terze parti. **PyOpenGL** è un'associazione Python allo standard OpenGL, che è una libreria grafica integrata con tutti i principali sistemi operativi. Ciò conferisce ai widget risultanti un aspetto e una sensazione nativi.

PyGlet è un altro collegamento Python al OpenGL librerie, ma può anche creare applicazioni GUI, che possono rendere la codifica utilizzando PyGlet più semplice rispetto all'utilizzo di PyOpenGL.

Ecco la panoramica dei moduli Python per questo capitolo:



PyOpenGL trasforma la nostra GUI

In questa ricetta, creeremo con successo una GUI Python che importi PyOpenGL moduli e funziona davvero! Per farlo, supereremo alcune sfide iniziali.

Questa ricetta mostrerà un modo provato che funziona. Se sperimenti da solo e rimani bloccato, ricorda le famose parole di Thomas A. Edison.



Thomas Edison, inventore della lampadina, ha risposto a una domanda di un giornalista che parlava dei fallimenti di Edison. Edison ha risposto: "*non ho fallito. Ho appena trovato 10.000 modi che non funzioneranno.*"

Per prima cosa, dobbiamo installare il PyOpenGL modulo di estensione. Dopo aver installato con successo il PyOpenGL moduli che corrispondono alla nostra architettura del sistema operativo, creeremo del codice di esempio.

Prepararsi

Installeremo il PyOpenGL pacchetto. In questo libro, utilizziamo il sistema operativo Windows 10 a 64 bit e Python 3.6. Lo screenshot dei download che segue è per questa configurazione.

Useremo anche wxPython. Se non hai wxPython installato, puoi leggere alcune ricette del capitolo precedente su come installare wxPython e come usare questo framework GUI.



Stiamo utilizzando la versione wxPython Phoenix, che è la versione più recente ed è destinata a sostituire la versione classica originale di wxPython in futuro.

Come farlo...

Per utilizzare PyOpenGL, dobbiamo prima installarlo. Il seguente URL è il sito Web ufficiale del programma di installazione del pacchetto Python:

<https://pypi.python.org/pypi/PyOpenGL/3.0.2>.

The screenshot shows the PyOpenGL 3.0.2 download page. At the top, it says "PyOpenGL 3.0.2" and "Standard OpenGL bindings for Python". A green button labeled "Downloads ↓" is visible. On the right, there's a sidebar with "Not Logged In" options like "Login", "Register", and "Login with Google". Below that is a "Status" section with "Nothing to report". The main area contains a table of files:

File	Type	Py Version	Uploaded on	Size
PyOpenGL-3.0.2.tar.gz (md5)	Source		2012-10-02	871KB
PyOpenGL-3.0.2.win-amd64.exe (md5)	MS Windows installer	any	2012-10-02	1MB
AMD64 Installer				
PyOpenGL-3.0.2.win32.exe (md5)	MS Windows installer	any	2012-10-02	1MB
PyOpenGL-3.0.2.zip (md5)	Source		2012-10-02	1MB

Questa sembra essere l'installazione corretta ma, a quanto pare, non funziona con il sistema operativo Windows 10 a 64 bit con Python 3.6 a 64 bit.

Un posto migliore per cercare i pacchetti di installazione di Python è stato menzionato in una ricetta nel capitolo precedente. Probabilmente lo conosci già. L'URL è <http://www.lfd.uci.edu/~gohlke/pythonlibs/>.

Dobbiamo scaricare il pacchetto che corrisponde sia al nostro sistema operativo che alla nostra versione Python. Viene fornito con il nuovo formato. Se stai usando una versione precedente di Python, devi installare Pythonruota pacchetto prima:



Come installare Python ruota pacchetto è descritto in una ricetta, *Installazione di Matplotlib usando pip con estensione whl*, nel Capitolo 5, *Grafici Matplotlib*.

Installazione PyOpenGL tramite il PyOpenGL-3.1.1-cp36-cp36m-win_amd64.whl file usando il pipè Il comando ha esito positivo e installa tutti i moduli a 64 bit richiesti. Sostituisci <il tuo percorso completo> con il percorso completo che hai scaricato il .quando installatore a:

pip install <il tuo percorso completo> PyOpenGL-3.1.1-cp36-cp36m-win_amd64.whl

Quando ora proviamo a importarne qualcuno PyOpenGL moduli, funziona, come si può vedere in questo esempio di codice:

```
importa wx
da wx import glcanvas da
OpenGL.GL import * from
OpenGL.GLU import *
```

Tutto ciò che sta facendo questo codice è importarne diversi OpenGL Moduli Python, oltre a importare wxPython. Non fa nient'altro ma, quando eseguiamo il nostro modulo Python, non riceviamo errori.

Ciò dimostra che abbiamo installato con successo il OpenGL collegamenti a Python.

Ora che il nostro ambiente di sviluppo è stato configurato con successo, possiamo provarlo utilizzando wxPython.



Molti esempi online sono limitati all'uso di Python 2 e alla versione classica di wxPython. Stiamo usando Python 3.6 e Phoenix.

L'utilizzo del codice basato sugli esempi demo di wxPython crea un cubo 3D funzionante. Al contrario, l'esecuzione dell'esempio del cono non ha funzionato, ma questo esempio ci ha fatto iniziare sulla strada giusta.

Ecco l'URL: <http://wiki.wxpython.org/GLCanvas%20update>

Ecco alcune modifiche al codice:

```
importa wx
da wx import glcanvas da
OpenGL.GL import * from
OpenGL.GLU import *

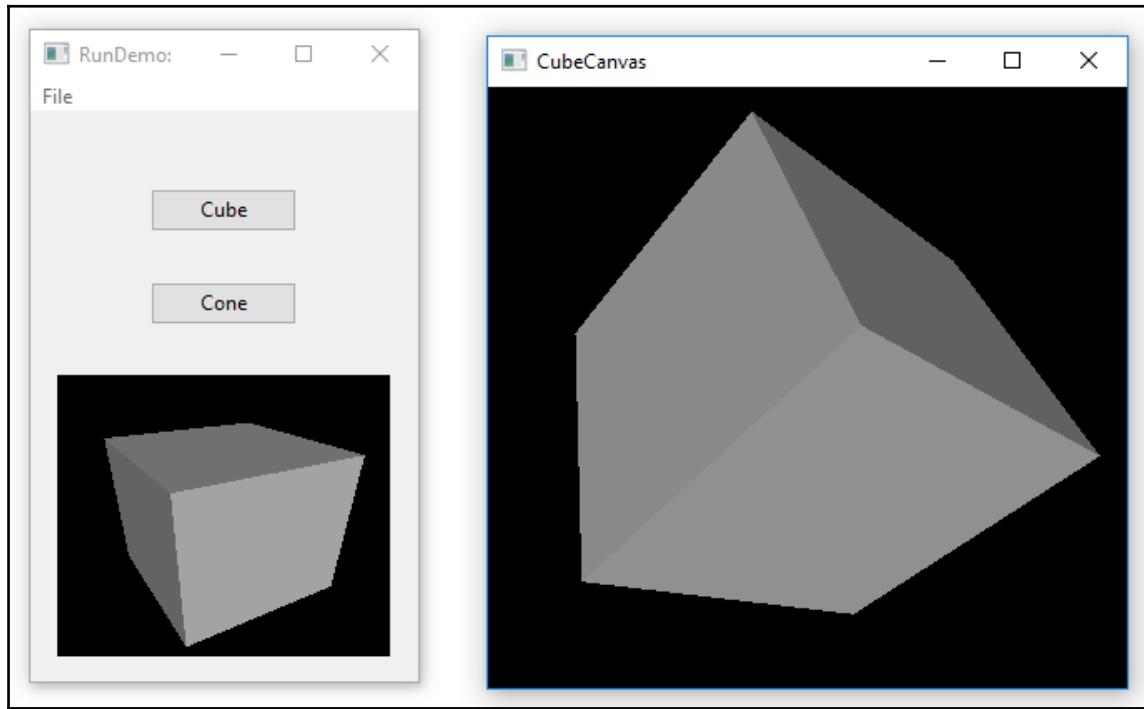
class MyCanvasBase(glcanvas.GLCamera):
    def __init__(self, genitore):
        glcanvas.GLCamera.__init__(self, genitore, -1)
        # Questo contesto mancava nel codice originale
```

```
self.context = glcanvas.GLContext(self)

def OnPaint(self, evento):
    dc = wx.PaintDC(self)
    # self.SetCurrent()                      # commentato perché:
    self.SetCurrent(self.context) # Dobbiamo passare in un contesto
```

Ora possiamo creare la seguente GUI:

```
import_OpenGL_cube_and_cone.py
```



Nella versione classica di wxPython, Imposta corrente() non richiedeva un contesto. Ecco del codice che potremmo trovare durante la ricerca online:

```
def OnPaint(self, evento):
    dc = wx.PaintDC(self)
    self.SetCurrent()
    se non self.init:
        self.InitGL()
        self.init = True
    self.OnDraw()
```

Il codice precedente non funziona quando si utilizza wxPython Phoenix. Possiamo cercare la sintassi corretta per Phoenix online:

The screenshot shows a web browser displaying the wxPython API documentation. The URL is https://wxpython.org/Phoenix/docs/html/wx.glccanvas.GLCanvas.html#wx.glccanvas.GLCanvas.SetCurrent. The page title is "wxPython | wx.glccanvas » wx.glccanvas.GLCanvas". On the left, there's a sidebar with the wxPython logo and a "Table Of Contents" section listing "wx.glccanvas.GLCanvas", "Class Hierarchy", "Methods Summary", and "Class API". The main content area contains the "SetCurrent(self, context)" method documentation. It includes a description: "Makes the OpenGL state that is represented by the OpenGL rendering context context current, i.e. it will be used by all subsequent OpenGL calls." Below this, it states: "This is equivalent to `wx.glcanvas.GLContext.SetCurrent` called with this window as parameter." To the right of the description, there are three boxes: "Parameters: context (`wx.glcanvas.GLContext`) –", "Return type: bool", and "Returns: False if an error occurred."

Come funziona...

In questa ricetta, abbiamo avuto le nostre prime esperienze di OpenGL con i collegamenti PyOpenGL Python. Sebbene OpenGL possa creare immagini davvero sorprendenti in vero 3D, ci siamo imbattuti in alcune sfide lungo la strada e poi abbiamo trovato soluzioni a queste sfide che lo hanno fatto funzionare.



Stiamo programmando in Python, creando immagini 3D!

La nostra GUI in 3D!

In questa ricetta, creeremo la nostra GUI usando wxPython. Riutilizzeremo parte del codice degli esempi demo di wxPython, che abbiamo ridotto al codice minimo richiesto per la visualizzazione OpenGL in 3D.



OpenGL è una biblioteca molto grande. Non entreremo in spiegazioni dettagliate di questa libreria. Ci sono molti libri e documentazione online disponibili se vuoi studiare OpenGL ulteriore. Ha il suo linguaggio di ombreggiatura.

Prepararsi

Leggere la ricetta precedente è probabilmente una buona preparazione per questa ricetta.

Come farlo...

Poiché l'intero codice Python è un po' lungo qui, mostreremo solo una parte del codice.

L'intero codice è disponibile online e questo modulo Python si chiama `wxPython_OpenGL_GUI.py`:

```
importa wx
da wx import glcanvas da
OpenGL.GL import * from
OpenGL.GLUT import *

# -----
class CanvasBase(glcanvas.GLCanvas):
    def __init__(self, genitore):
        glcanvas.GLCanvas.__init__(self, parent, -1) self.context =
        glcanvas.GLContext(self) self.init = False

        # Cube 3D avvia la rotazione
        self.last_X = self.x = 30 self.last_Y
        = self.y = 30

        self.Bind(wx.EVT_SIZE, self.sizeCallback) self.Bind(wx.EVT_PAINT,
        self.paintCallback) self.Bind(wx.EVT_LEFT_DOWN,
        self.mouseDownCallback) self.Bind(wx.EVT_LEFT_UP,
        self.mouseUpCallback) self. Bind(wx.EVT_MOTION,
        self.mouseMotionCallback)

    def sizeCallback(self, event):
        wx.CallAfter(self.setViewport)
        evento.Skip()

    def setViewport(self):
        self.size = self.GetClientSize()
        self.SetCurrent(self.context)
        glViewport(0, 0, self.size.width, self.size.height)

    def paintCallback(self, evento):
        wx.PaintDC(self)
        self.SetCurrent(self.context) if not
        self.init:
```

```
        self.initGL()
        self.init = True
        self.onDraw()

def mouseDownCallback(self, event):
    self.CaptureMouse()
    self.x, self.y = self.last_X, self.last_Y = event.GetPosition()

def mouseUpCallback(self, evt):
    self.ReleaseMouse()

def mouseMotionCallback(self, evt):
    if evt.Dragging() & evt.LeftIsDown():
        self.last_X, self.last_Y = self.x, self.y
        self.x, self.y =
            evt.GetPosition()
        self.Refresh(False)

# -----
classe CubeCanvas(CanvasBase):
    def initGL(self):
        # imposta la proiezione di visualizzazione
        glMatrixMode(GL_PROJECTION) glFrustum(-0.5, 0.5,
-0.5, 0.5, 1.0, 3.0)

        # visualizzatore di posizione
        glMatrixMode(GL_MODELVIEW)
        glTranslatef(0.0, 0.0, -2.0)

        # posizione oggetto
        glRotatef(self.y, 1.0, 0.0, 0.0)
        glRotatef(self.x, 0.0, 1.0, 0.0)

        glEnable(GL_DEPTH_TEST)
        glEnable(GL_LIGHTING)
        glEnable(GL_LIGHT0)

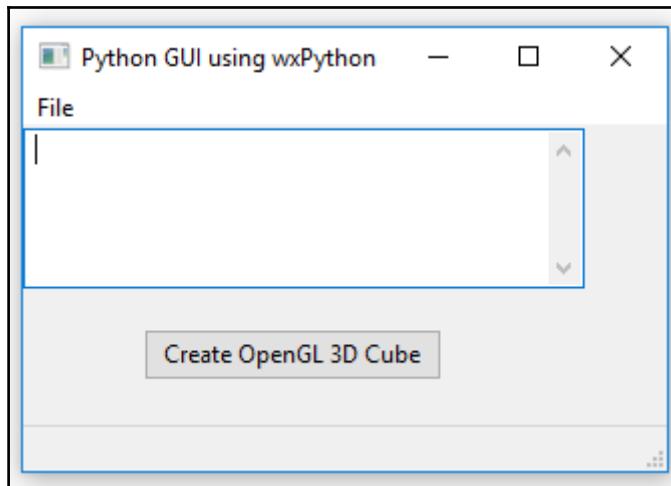
    def onDraw(self):
        # colore chiaro e buffer di profondità
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)

        # disegna sei facce di un cubo
        glBegin(GL_QUADS)
        glNormal3f( 0.0, 0.0, 1.0)
        glVertex3f( 0.5, 0.5, 0.5)
        glVertex3f(-0.5, 0.5, 0.5)
        glVertex3f(-0.5,-0.5, 0.5)
        glVertex3f( 0.5,-0.5, 0.5)
```

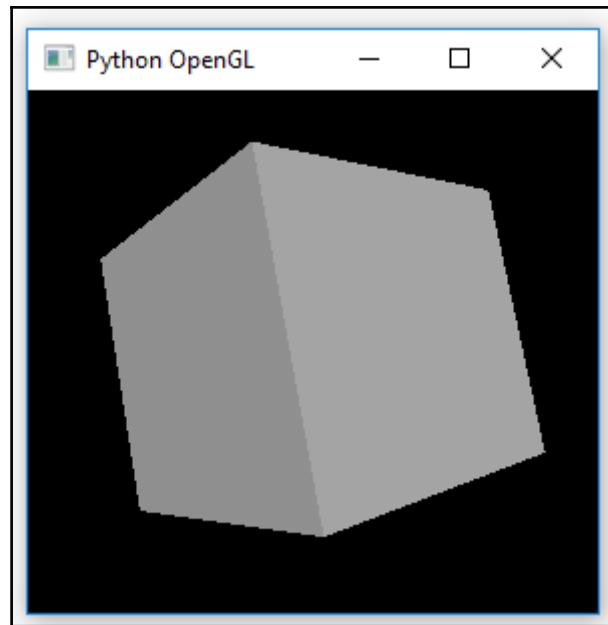
```
glNormal3f(0.0,0.0,-1.0)
glVertex3f(-0.5,-0.5,-0.5)
```

```
# ===== app =
wx.App()
frame = wx.Frame(Nessuno, , size=(300,230))
GUI(frame)
cornice.Mostra()
app.MainLoop()
```

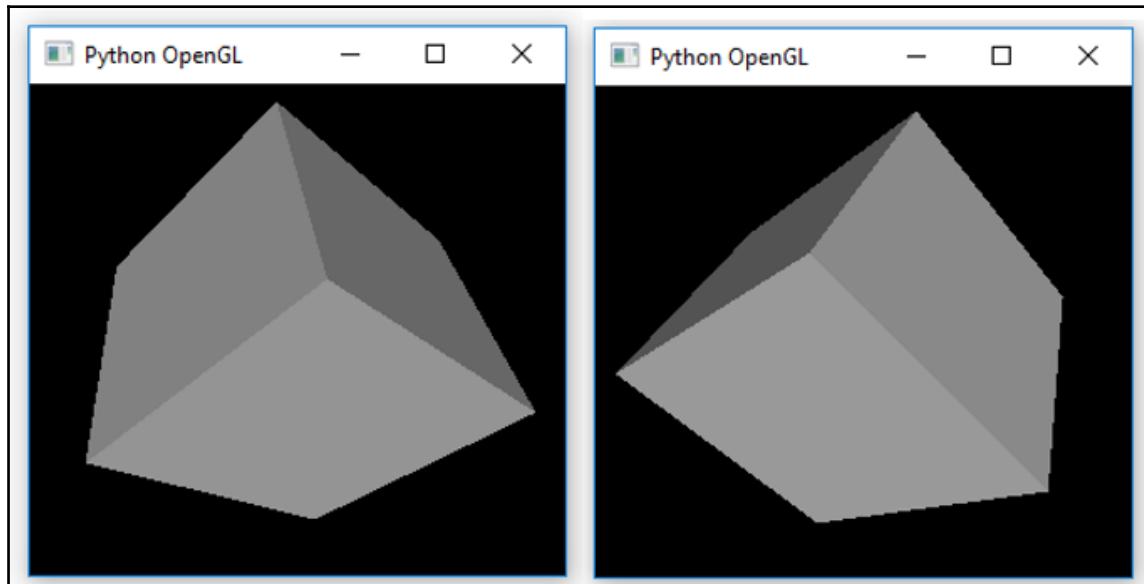
La seguente schermata mostra la nostra GUI wxPython: `wxPython_OpenGL_GUI.py`



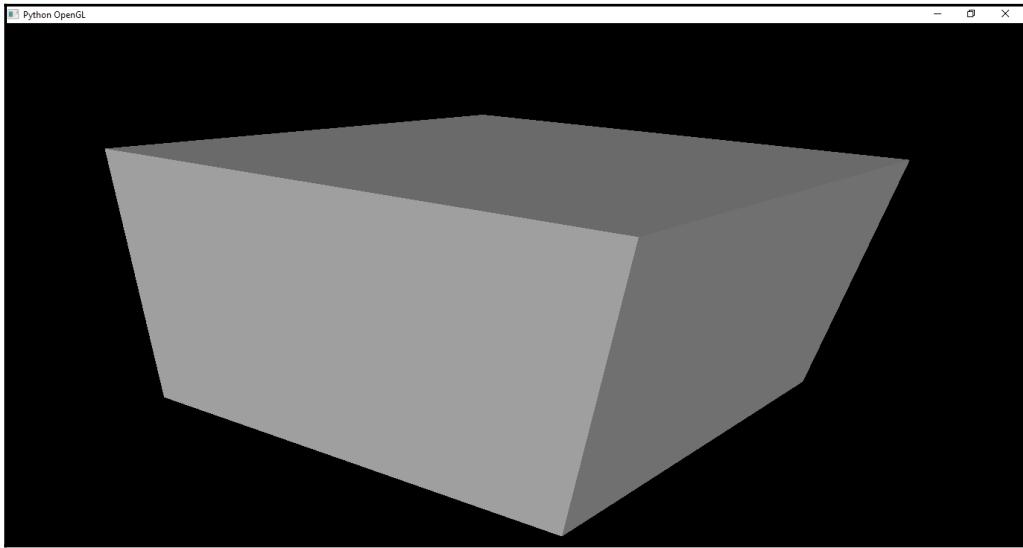
Quando facciamo clic sul pulsante widget, viene visualizzata la seguente seconda finestra:



Ora possiamo usare il mouse per trascinare il cubo per vedere tutti i suoi sei lati:



Possiamo anche massimizzare questa finestra e le coordinate scaleranno, e possiamo ruotare questo cubo in questa finestra molto più grande!



Il cubo potrebbe anche essere un'astronave di Star Trek! Dobbiamo solo diventare un programmatore avanzato in questa tecnologia se questo è ciò che vogliamo sviluppare.

Molti videogiochi vengono sviluppati utilizzando OpenGL.



Come funziona...

Per prima cosa abbiamo creato una normale GUI wxPython e vi abbiamo inserito un widget pulsante. Facendo clic su questo pulsante si richiama l'importazioneOpenGL Librerie 3D. Il codice utilizzato fa parte degli esempi demo di wxPython, che abbiamo leggermente modificato per far funzionare il codice con Phoenix.



Questa ricetta ha incollato la nostra GUI a questa libreria.

OpenGL è una biblioteca così grande e molto impressionante. Questa ricetta ha dato un assaggio di come creare un esempio funzionante in Python.



Spesso, un esempio funzionante è tutto ciò di cui abbiamo bisogno per iniziare il nostro viaggio.

Usare le bitmap per rendere carina la nostra GUI

Questa ricetta è stata ispirata da un framework di creazione di IDE wxPython che, a un certo punto, funzionava. Riutilizzeremo un'immagine bitmap dalla grande quantità di codice fornita da questo progetto.

L'URL del repository GitHub è <https://github.com/reingart/gui2py>.

Non sono stato in grado di ricreare questo IDE usando Python 3.6. Eppure mostra ciò che è possibile. Se sei interessato a sviluppare un framework IDE drag and drop, puoi aiutare l'intera comunità di sviluppatori Python Windows (e competere con MS Visual Studio...) facendo funzionare questo codice su Python 3.6 e oltre.

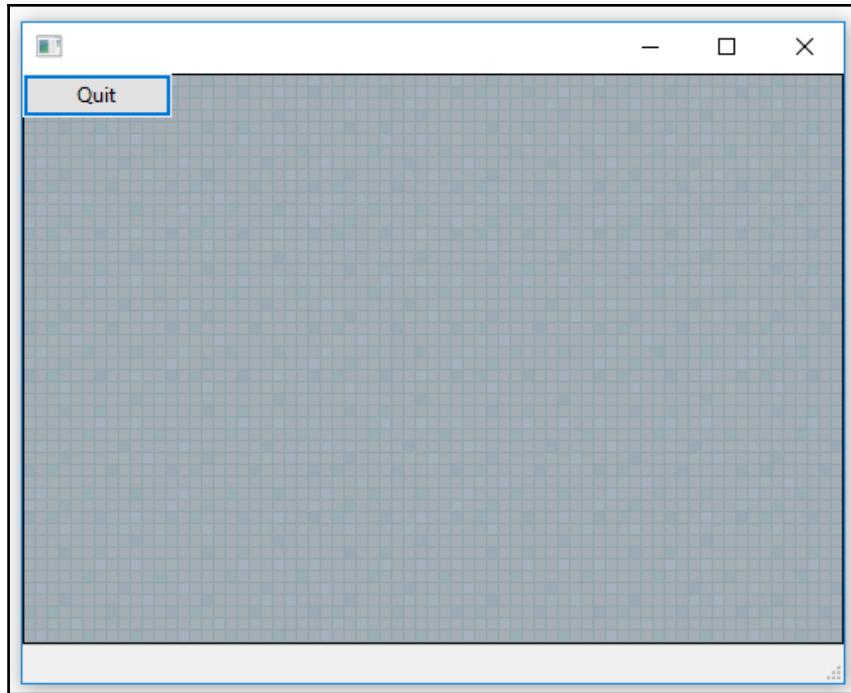
Prepararsi

Continueremo a utilizzare wxPython in questa ricetta, quindi leggere almeno parti del capitolo precedente potrebbe essere utile come preparazione per questa ricetta.

Come farlo...

Dopo il reverse engineering del gui2py codice e apportando altre modifiche a questo codice, possiamo ottenere il seguente widget di finestra, che mostra uno sfondo piacevole e piastrellato:

wxPython_Wallpaper_simple.py



Ovviamente, abbiamo perso molti widget durante il refactoring del codice dal sito Web di cui sopra, ma ci fornisce uno sfondo interessante e facendo clic sul **Smettere** il pulsante funziona ancora

Il prossimo passo è capire come integrare la parte interessante del codice nella nostra GUI. Lo facciamo aggiungendo il seguente codice alla GUI nella ricetta precedente:

```
# -----
class GUI(wx.Panel): # Sottoclasse wxPython Panel
    def __init__(self, genitore):
        wx.Panel.__init__(self, genitore)

        imageFile = 'Tile.bmp'
        self.bmp = wx.Bitmap(imageFile)
        # reagire a un evento di ridimensionamento e ridisegnare l'immagine
        parent.Bind(wx.EVT_SIZE, self.canvasCallback)
```

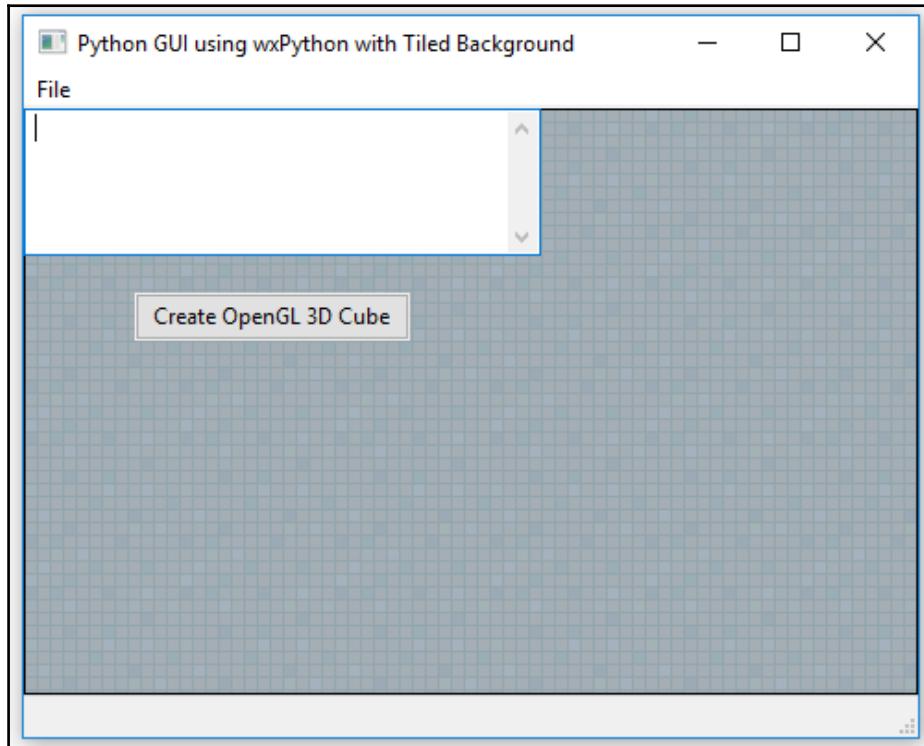
```
def canvasCallback(self, event=Nessuno):
    # crea il contesto del dispositivo
    dc = wx.ClientDC(self)
    brushBMP = wx.Brush(self.bmp)
    dc.SetBrush(brushBMP)
    larghezza, altezza = self.GetClientSize()
    dc.DrawRectangle(0, 0, larghezza, altezza)
```



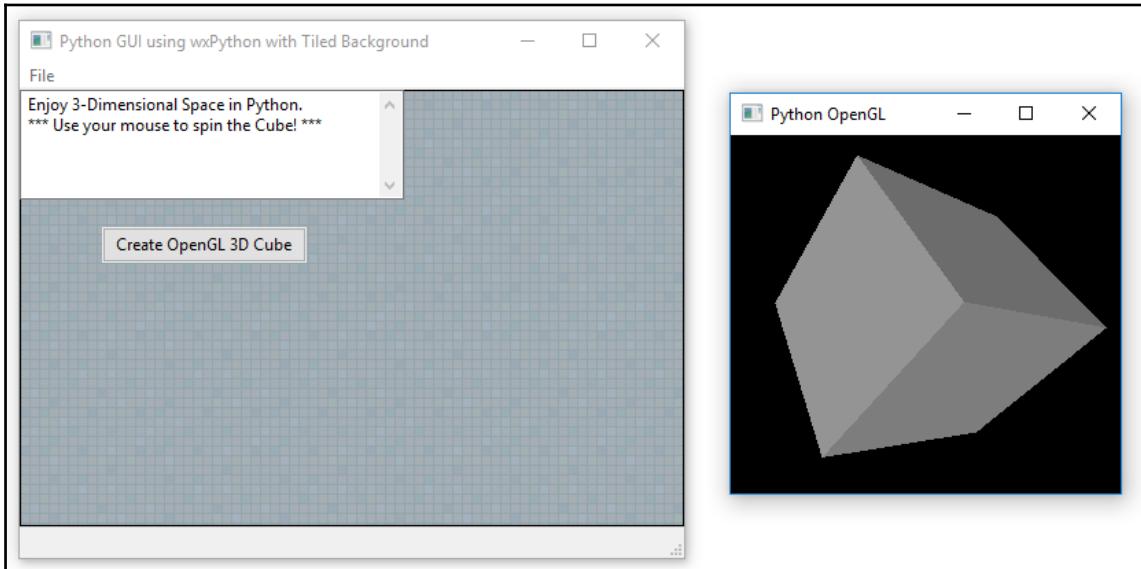
Dobbiamo legarci al genitore, non al sé; in caso contrario, la nostra bitmap non verrà visualizzata.

L'esecuzione del nostro codice migliorato ora affianca una bitmap come sfondo della nostra GUI:

`wxPython_Wallpaper.py`



Facendo clic sul pulsante si richiama ancora il nostro OpenGL Disegno 3D, quindi non abbiamo perso alcuna funzionalità:



Come funziona...

In questa ricetta, abbiamo migliorato la nostra GUI utilizzando una bitmap come sfondo. Abbiamo affiancato l'immagine bitmap e, quando abbiamo ridimensionato la finestra della GUI, la bitmap si è adattata automaticamente per riempire l'intera area della tela su cui stavamo dipingendo utilizzando il contesto del dispositivo.

Il precedente codice wxPython può caricare diversi formati di file immagine.



PyGlet trasforma la nostra GUI più facilmente di PyOpenGL

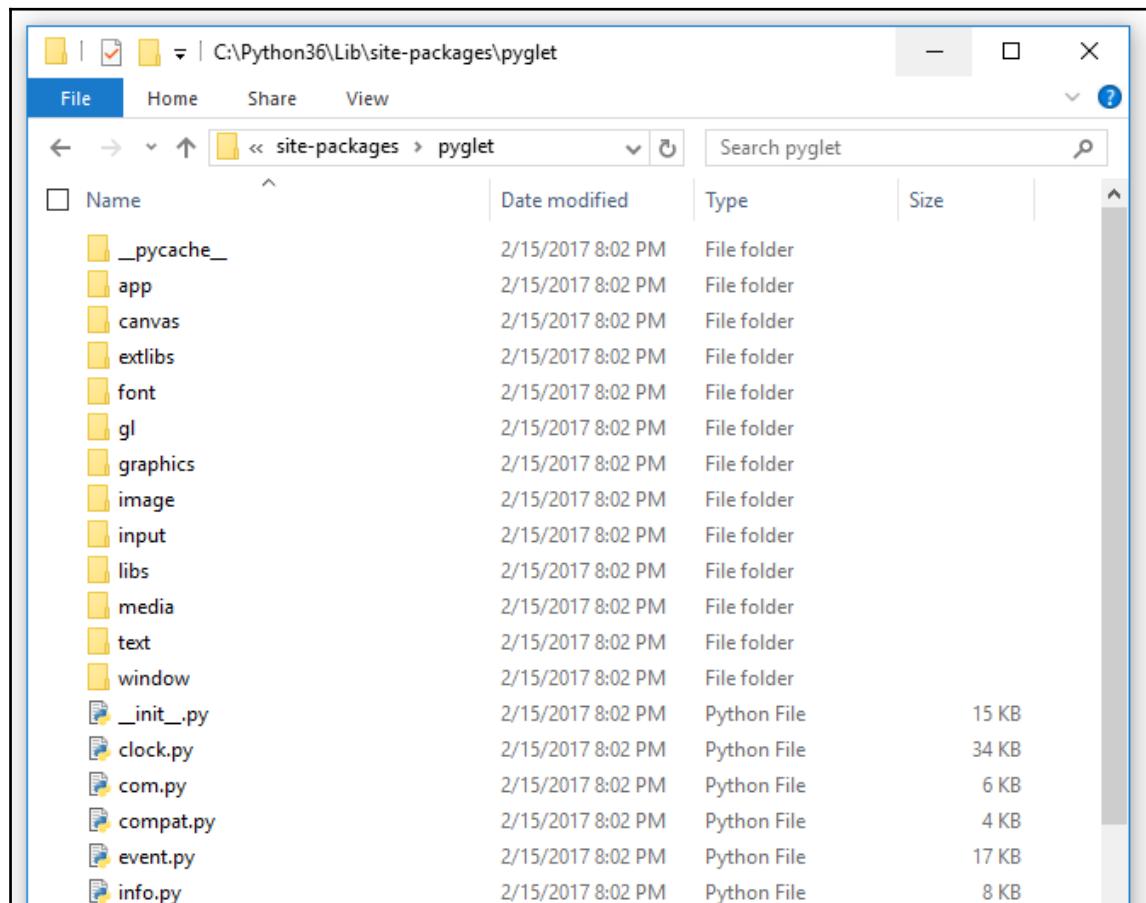
In questa ricetta, utilizzeremo il framework di sviluppo della GUI PyGlet per creare le nostre GUI.

PyGLet è più facile da usare rispetto a PyOpenGL, poiché viene fornito con il proprio ciclo di eventi della GUI, quindi non è necessario utilizzare tkinter o wxPython per creare la nostra GUI.

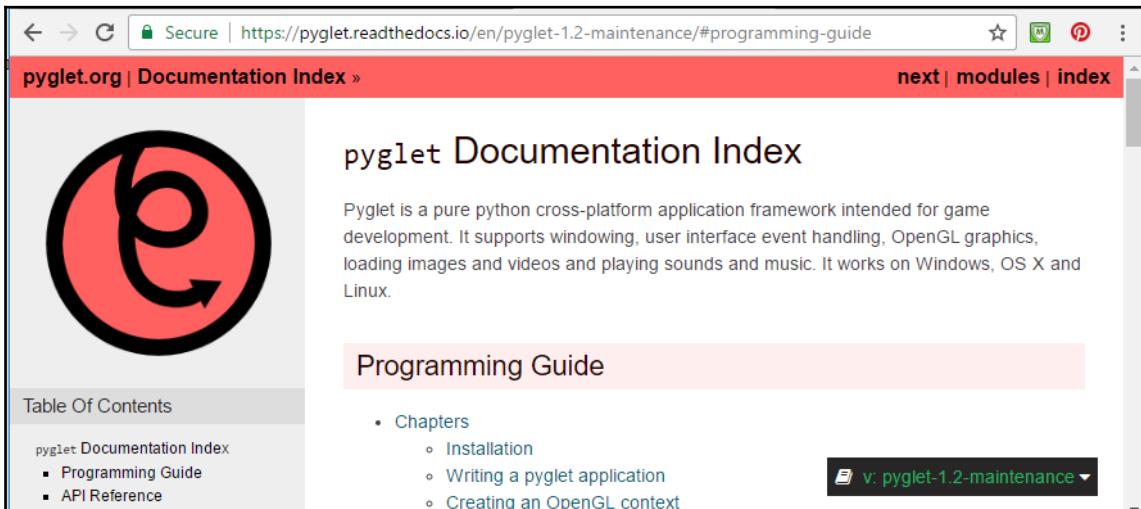
Come farlo...

Per utilizzare PyGLet, dobbiamo prima installare questo plugin Python di terze parti.

Usando il pipè comando, possiamo installare facilmente la libreria e un'installazione riuscita appare come questa nel nostro pacchetti-sito Cartella Python:



La documentazione online si trova all'indirizzo <https://pyglet.readthedocs.org/en/pyglet-1.2-maintenance/> sito web per la versione corrente:



pyglet Documentation Index

Pyglet is a pure python cross-platform application framework intended for game development. It supports windowing, user interface event handling, OpenGL graphics, loading images and videos and playing sounds and music. It works on Windows, OS X and Linux.

Programming Guide

- Chapters
 - Installation
 - Writing a pyglet application
 - Creating an OpenGL context

v: pyglet-1.2-maintenance

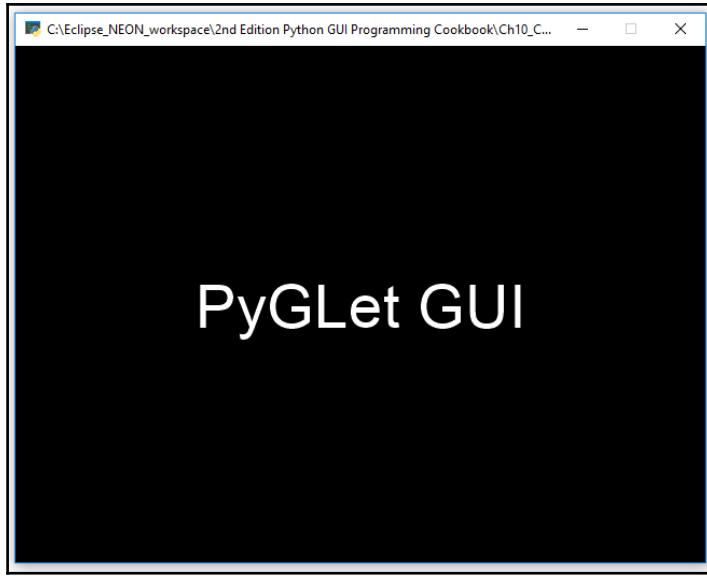
Una prima esperienza con il porcellino libreria può avere il seguente aspetto:

```
importa pigiama
finestra = pyglet.window.Window()
label = pyglet.text.Label('PyGlet GUI',
                         font_size=42,
                         x=window.width//2, y=window.height//2,
                         anchor_x='center', anchor_y='center')
@finestra.evento
def on_draw():
    finestra.clear()
    etichetta.disegna()

pyglet.app.run()
```

Il codice precedente è del funzionario pyglet.org sito Web e risultati nella seguente GUI completamente funzionante:

`pyglet_GUI_Simple.py`



Come funziona...

In questa ricetta, abbiamo utilizzato un altro modulo Python di terze parti che avvolge il OpenGL biblioteca.

Questa libreria è dotata di una propria potenza di elaborazione del ciclo di eventi, che ci consente di evitare di dover fare affidamento su un'altra libreria per creare una GUI Python in esecuzione.

Abbiamo esplorato il sito Web ufficiale, che ci mostra come installare e utilizzare questa fantastica libreria GUI.

La nostra GUI in fantastici colori

In questa ricetta, estenderemo la nostra GUI scritta usando PyGlet dalla ricetta precedente, *PyGlet trasforma la nostra GUI più facilmente di PyOpenGL*, trasformandolo in vero 3D.

Aggiungeremo anche alcuni colori fantasiosi. Questa ricetta è ispirata a un codice di esempio dal *SuperBibbia OpenGL* serie di libri. Crea un cubo molto colorato, che possiamo girare in uno spazio tridimensionale usando i pulsanti della tastiera su, giù, sinistra e destra.

Abbiamo leggermente migliorato il codice di esempio facendo ruotare l'immagine quando si tiene premuto uno dei tasti invece di dover premere e rilasciare il tasto.

Prepararsi

La ricetta precedente, *PyGlet trasforma la nostra GUI più facilmente di PyOpenGL*, spiega come installare PyGlet e fornisce un'introduzione a questa libreria. Se non lo hai fatto, probabilmente è una buona idea sfogliare quella ricetta.



Nella documentazione in linea, PyGlet è solitamente scritto tutto in minuscolo. Anche se questo potrebbe essere un modo Pythonic, mettiamo in maiuscolo la prima lettera di una classe e usiamo il minuscolo per i nomi di variabili, metodi e funzioni per iniziare ogni nome.

Come farlo...

Il codice seguente crea il cubo colorato 3D, che segue il codice. Questa volta utilizzeremo i tasti freccia della tastiera per ruotare l'immagine invece del mouse:

```
importa pigiama
da pyglet.gl import *
da pyglet.window import chiave da
OpenGL.GLUT import *

FINESTRA      = 400
INCREMENTO = 5

class Window(pyglet.window.Window):

    # Cube 3D avvia la rotazione
    xRotation = yRotation = 30

    def __init__(self, larghezza, altezza, titolo=""):
        super(Window, self).__init__(width, height, title) glClearColor(0, 0, 0,
        1)
        glEnable(GL_DEPTH_TEST)

    def on_draw(self):
        # Cancella la finestra GL corrente
        self.clear()

        # Spingi Matrix nello stack
        glPushMatrix()
```

```
glRotatef(self.xRotation, 1, 0, 0)
glRotatef(self.yRotation, 0, 1, 0)

# Disegna i sei lati del cubo
glBegin(GL_QUADS)

    # Bianca
    glColor3ub(255, 255, 255)
    glVertex3f(50,50,50)

    # Giallo
    glColor3ub(255, 255, 0)
    glVertex3f(50,-50,50)

    # Rosso
    glColor3ub(255, 0, 0)
    glVertex3f(-50,-50,50)
    glVertex3f(-50,50,50)

    # Blu
    glColor3f(0, 0, 1)
    glVertex3f(-50,50,-50)

# <... più colore definisce per le facce del cubo>

glEnd()

# Pop Matrix fuori dallo stack
glPopMatrix()

def on_resize(self, larghezza, altezza):
    # imposta il Viewport
    glViewport(0, 0, larghezza, altezza)

    # utilizzando la modalità di proiezione
    glMatrixMode(GL_PROJECTION)
    glLoadIdentity()

    aspectRatio = larghezza / altezza
    gluPerspective(35,
                   aspectRatio, 1, 1000)

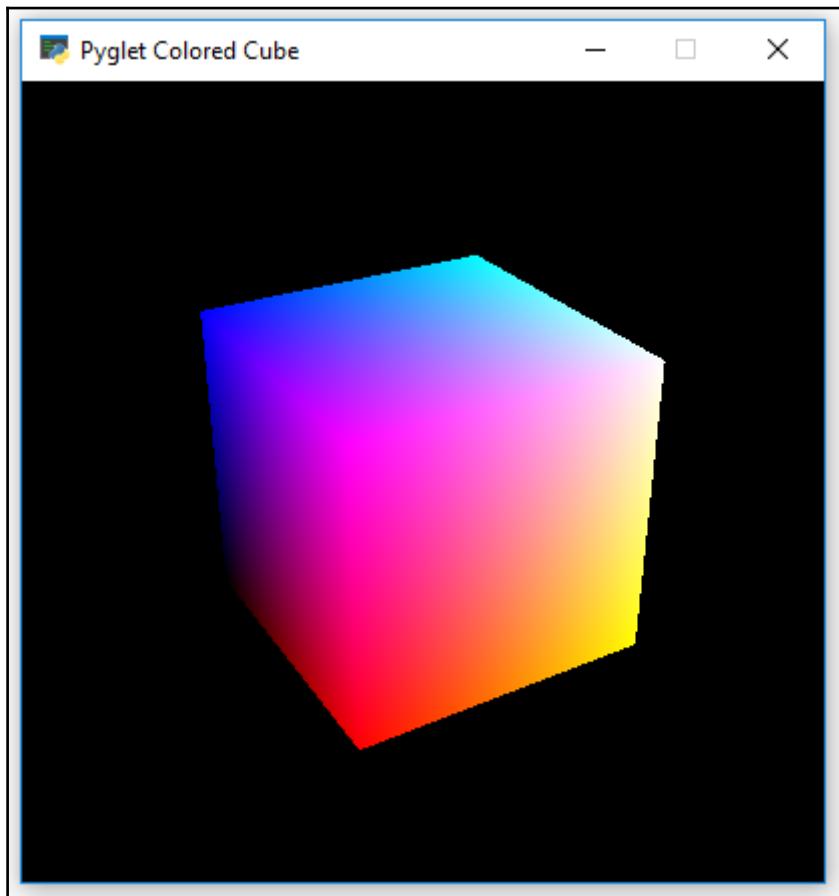
    glMatrixMode(GL_MODELVIEW)
    glLoadIdentity()
    glTranslatef(0, 0, -400)

def on_text_motion(self, motion):
    if motion == key.UP:
        self.xRotation -= INCREMENT
```

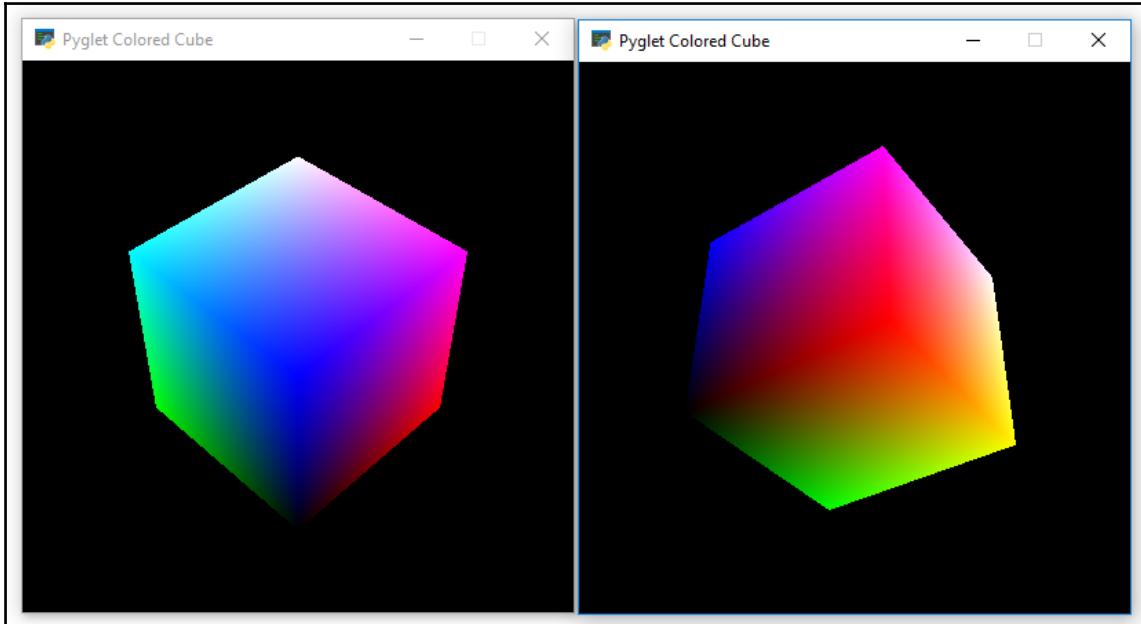
```
        elif motion == key.DOWN:
            self.xRotation += INCREMENT
        elif motion == key.LEFT:
            self.yRotation -= INCREMENT
        elif motion == key.RIGHT:
            self.yRotation += INCREMENT

    if __name__ == '__main__':
        Window(WINDOW, WINDOW, 'Pyglet Colored Cube')
        pyglet.app.run()
```

pyglet_GUI.py



Usando i tasti freccia della tastiera, possiamo far girare il cubo 3D intorno a:



Come funziona...

In questa ricetta, abbiamo usato porcellino per creare un cubo colorato, che possiamo ruotare nello spazio 3D usando i tasti freccia della tastiera.

Abbiamo definito diversi colori per le sei facce del nostro cubo e abbiamo usato porcellino per creare la nostra cornice della finestra principale.

Il codice è simile a una ricetta precedente in questo capitolo, in cui abbiamo usato il wxPython libreria per creare un cubo. La ragione di ciò è che, sotto il cofano, sia wxPython che PyGLet usano ilOpenGL biblioteca.

Animazione OpenGL

Nelle ricette precedenti, abbiamo utilizzato i framework wxPython e PyGLet per visualizzare OpenGL. In questa ricetta utilizzeremo puro Python e OpenGL per creare un rettangolo rosso che rimbalza all'interno di una finestra con sfondo blu.



Gli esempi in questa ricetta sono stati tradotti dal linguaggio di programmazione C in Python usando il *OpenGL SuperBible Quarta Edizione* come una guida. Sebbene la quarta edizione sia stata pubblicata nell'anno 2007, gli esempi OpenGL funzionano ancora e possiamo usarli con Python.

Prepararsi

Questa ricetta richiede il PyOpenGL pacchetto. La prima ricetta di questo capitolo, *PyOpenGL trasforma la nostra GUI*, spiega come installare questo pacchetto.

Come farlo...

Innanzitutto, importiamo diversi pacchetti da OpenGL. Nel principale() Funzione Python, inizializziamo il Utilità GL (eccesso) biblioteca. Quindi scegliamo un singolo buffer (al contrario del doppio buffering) e selezioniamo anche il**Rosso, Verde, Blu, Alfa (RGBA)** modalità colore.

Creiamo quindi una finestra tramite GLUT. Non c'è bisogno di tkinter, wxPython o PyGLet. Possiamo farlo direttamente tramiteOpenGL GLUT funzioni.



In Python 3.6, dobbiamo passare la stringa del titolo della finestra come byte. Lo facciamo inserendo **ab** davanti alla stringa, che codifica la stringa come byte.

Successivamente, creiamo e definiamo una funzione di visualizzazione, RenderScene(). Questa funzione viene chiamata quando avviamo il ciclo di eventi di Windows glut.

Nel SetupRC() funzione, definiamo un colore da utilizzare per cancellare la finestra di rendering/visualizzazione. I valori validi sono numeri float compresi tra 0.0 e 1.0. Il passaggio di valori diversi in RGB crea quindi colori diversi. Nell'esempio seguente, definiamo il colore blu.

Mentre abbiamo posizionato il SetupRC() funzione sotto glutDisplayFunc(), questa funzione in realtà viene chiamata per prima perché RenderScene() viene chiamato solo quando chiamiamo il glutMainLoop() funzione.

Ecco il codice:

`OpenGL_SuperBible_Simple.py`

```
⚠ from OpenGL.GLUT import *
⚠ from OpenGL.GL import *
⚠ from OpenGL.GLU import *

def RenderScene():                      # display callback function
    glClear(GL_COLOR_BUFFER_BIT)         # clear window with color defined in SetupRC
    glFlush()                           # flush/execute the OpenGL drawing command(s)

def SetupRC():                          # rendering context
    glClearColor(0.0, 0.0, 1.0, 1.0)     # RGBA: R=0.0, G=0.0, B=1.0 => becomes Blue

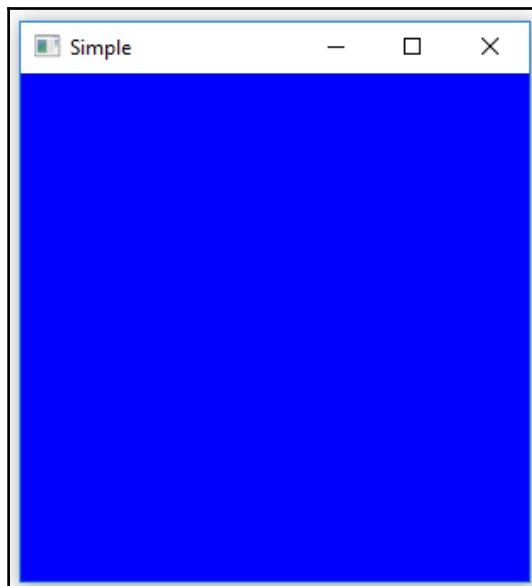
def main():
    glutInit()
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA)      # single buffer; RGBA color mode
    glutCreateWindow(b"Simple")                         # Python 3: bytes instead of string for Title
    glutDisplayFunc(RenderScene)
    SetupRC()

    glutMainLoop()

=====
main()
```

L'esecuzione del codice crea la seguente finestra, utilizzando puro Python con OpenGL:

OpenGL_SuperBible_Simple.py



Per visualizzare un componente del disegno all'interno di questo OpenGL finestra, miglioriamo il RenderingScene() funzione creando un rettangolo quadrato con una dimensione di 25 x 25 pixel.

Aggiungiamo anche una nuova funzione, Cambia dimensione(), che viene chiamato automaticamente dal GLUT libreria ogni volta che la finestra viene ridimensionata o ridisegnata:

OpenGL_SuperBible_Simple_Rectangle.py

```
⚠ from OpenGL.GLUT import *
⚠ from OpenGL.GL import *
⚠ from OpenGL.GLU import *

def RenderScene():                      # display callback function
    glClear(GL_COLOR_BUFFER_BIT)          # clear window with color defined in SetupRC
    #   R   G   B
    glColor3f(1.0, 0.0, 0.0)             # set drawing color to Red
    glRectf(-25.0, 25.0, 25.0, -25.0)   # function expects 3 floats
    # draw a filled rectangle with above color
    glFlush()                           # flush/execute the OpenGL drawing command(s)

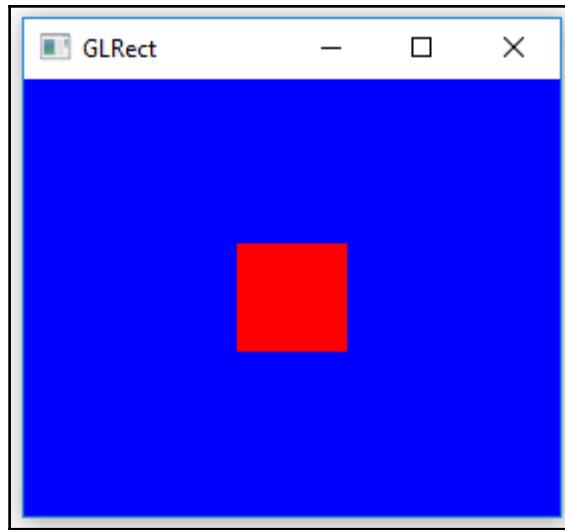
def SetupRC():                          # rendering context setup
    glClearColor(0.0, 0.0, 1.0, 1.0)     # RGBA: R=0.0, G=0.0, B=1.0=white => becomes Blue

def ChangeSize(w, h):                  # callback when window size changes
    if h == 0: h = 1
    glViewport(0, 0, w, h)              # set Viewport to Window dimensions
    glMatrixMode(GL_PROJECTION)        # define the viewing volume
    glLoadIdentity()                  # reset coordinate system
    aspectRatio = GLfloat(w).value / GLfloat(h).value # establish clipping volume
    ....                                # GLfloat becomes ctypes.c_float
    ....                                # Use the value attribute before dividing
    if w <= h: glOrtho(-100.0, 100.0, -100.0 / aspectRatio, 100.0 / aspectRatio, 1.0, -1.0)
    else:    glOrtho(-100.0 * aspectRatio, 100.0 * aspectRatio, -100.0, 100.0, 1.0, -1.0)
    glMatrixMode(GL_MODELVIEW)
    glLoadIdentity()

def main():
    glutInit()
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA)      # single buffer; RGBA color mode
    glutCreateWindow(b"GLRect")                        # Python 3: bytes instead of string for Title
    glutDisplayFunc(RenderScene)
    glutReshapeFunc(ChangeSize)
    SetupRC()
    glutMainLoop()
=====
main()
```

Ecco la finestra risultante:

OpenGL_SuperBible_Simple_Rectangle.py



Successivamente, animeremo il nostro quadrato rosso facendolo muovere attraverso la finestra, rimbalzando su ogni angolo. Per fare ciò, creiamo prima alcune variabili globali a livello di modulo:

OpenGL_SuperBible_Animation.py

```
⚠️ from OpenGL.GLUT import *
⚠️ from OpenGL.GL import *
⚠️ from OpenGL.GLU import *

# initial position and size
x1 = GLfloat(0.0).value
y1 = GLfloat(0.0).value
rect_size = GLfloat(25.0).value

# number of pixels to move each step
xstep = GLfloat(1.0).value
ystep = GLfloat(1.0).value

# initialize bouncing window
windowWidth  = GLfloat(133).value          # 800/600 = 1.33
windowHeight = GLfloat(100).value
```

Nel RenderScene() funzione, ora usiamo il doppio buffer:

```
def RenderScene():          # display callback function
    glClear(GL_COLOR_BUFFER_BIT)      # clear window with color defined in SetupRC
    #           R   G   B           # set drawing color to Red
    glColor3f(1.0, 0.0, 0.0)        # functions expects 3 floats
    glRectf(x1, y1, x1 + rect_size, y1 - rect_size) # draw a filled rectangle with above color

    glutSwapBuffers()             # flush and swap double buffers
```

Dobbiamo cambiare leggermente il nostro principale() passare GLUT_DOUBLE dentro glutInitDisplayMode() funzione:

```
def main():
    glutInit()
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB)      # Double buffer
    glutInitWindowSize(800, 600)                      # window size
    glutCreateWindow(b"Bouncing Red Square")         # Python 3: bytes instead of string for Title
    glutDisplayFunc(RenderScene)
    glutReshapeFunc(ChangeSize)
    glutTimerFunc(33, TimerFunction, 1)

    SetupRC()

    glutMainLoop()

=====
main()
```

Creiamo il Funzione Timer() essere passato in glutTimerFunc(). In questa funzione, facciamo un po' di matematica per posizionare il quadrato rosso e farlo rimbalzare sulle pareti della finestra. Alla fine della funzione, chiamiamo ricorsivamente questa funzione in glutTimerFunc():

```
def TimerFunction(value):
    global x1, xstep, y1, ystep

    # reverse direction left/right
    if ((x1 > windowHeight - rect_size) or (x1 < -windowWidth)):
        xstep = -xstep
    .....

    # reverse direction top/bottom
    if ((y1 > windowHeight) or (y1 < -windowHeight + rect_size)):
        ystep = -ystep
    .....

    # move the red square
    x1 += xstep
    y1 += ystep

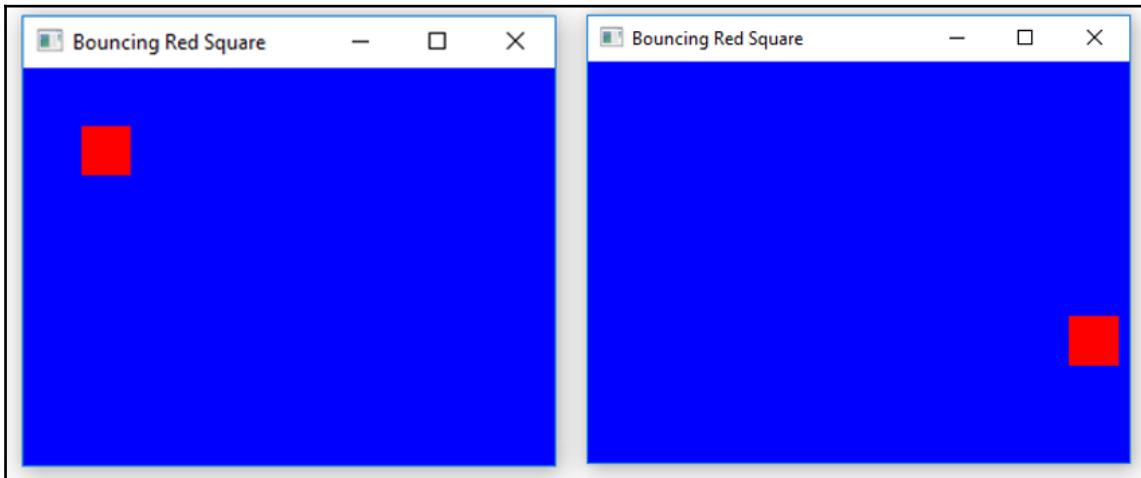
    # check the bounds of the clipping area
    if (x1 > (windowWidth - rect_size + xstep)):
        x1 = windowHeight - rect_size - 1
    elif (x1 < -(windowWidth + xstep)):
        x1 = -windowWidth - 1
    .....

    if (y1 > (windowHeight + ystep)):
        y1 = windowHeight - 1
    elif (y1 < -(windowHeight - rect_size + ystep)):
        y1 = -windowHeight + rect_size - 1
    .....

    # redraw the scene
    glutPostRedisplay()
    glutTimerFunc(33, TimerFunction, 1)      # recursive call
```

Quando ora eseguiamo il programma, il quadrato rosso si sposterà all'interno della finestra blu, rimbalzando su ogni angolo:

OpenGL_SuperBible_Animation.py



Come funziona...

In questa ricetta abbiamo usato il OpenGL librerie direttamente da Python. Abbiamo creato la nostra finestra senza dover utilizzare nessun altro framework GUI Python. Abbiamo esplorato alcuni dei tanti OpenGL funzioni e programmi di lavoro creati.



Una buona, breve introduzione a OpenGL e come usarlo da Python can essere trovato a <https://noobtuts.com/python/opengl-introduction>.

Creare una presentazione usando tkinter

In questa ricetta, creeremo una bella GUI per la presentazione di diapositive funzionante usando Python puro. Vedremo le limitazioni che hanno i built-in di base di Python, e poi esploreremo un altro modulo di terze parti disponibile chiamatoCuscino, che estende la funzionalità integrata di tkinter per quanto riguarda l'elaborazione delle immagini.

Mentre il nome Cuscino potrebbe sembrare un po' strano all'inizio, in realtà ha molta storia alle spalle.



In questo libro utilizziamo solo Python 3.6 e versioni successive. Non torneremo a Python 2. Guido ha espresso la sua decisione di rompere intenzionalmente la retrocompatibilità e ha deciso che Python 3 è il futuro della programmazione Python.

Per GUI e immagini, la vecchia linea di Python 2 ha un modulo molto potente chiamato PIL, che sta per Python Image Library. Questa libreria è dotata di una grande quantità di funzionalità che, diversi anni dopo la creazione di grande successo di Python 3, non è stata tradotta per Python 3.

Molti sviluppatori scelgono ancora di utilizzare Python 2 invece del futuro, come progettato dal *Benevolo dittatore di Python* perché Python 2 ha ancora più librerie disponibili.

Questo è un po' triste. Fortunatamente, è stata creata un'altra libreria di immagini per funzionare con Python 3.6 e si chiama PIL più qualcosa.

Pillow non è compatibile con la libreria PIL Python 2.



Prepararsi

Nella prima parte di questa ricetta utilizzeremo puro Python. Per migliorare il codice, installeremo un altro modulo Python usando il usingpipì funzionalità, quindi mentre è molto probabile che tu abbia familiarità con pipa, un po' di conoscenza di come usarlo potrebbe essere utile.

Come farlo...

Innanzitutto, creeremo una GUI funzionante che mischia le diapositive all'interno di una cornice di finestra utilizzando puro Python. Ecco il codice funzionante e alcuni screenshot dei risultati dell'esecuzione di questo codice:

```
from tkinter import Tk, PhotoImage, Label from  
itertools import cycle  
da os import listdir
```

Presentazione di classe (Tk):

```
# eredita il framework della GUI che estende tkinter  
def __init__(self, msShowTimeBetweenSlides=1500):
```

```
# inizializza la super classe tkinter
Tk.__init__(self)

# volta che verrà mostrata ogni diapositiva
self.showTime = msShowTimeBetweenSlides

# cerca le immagini nella directory di lavoro corrente
listOfSlides = [diapositiva per diapositiva in listdir()
                if slide.endswith('gif')]

# ciclo diapositive da mostrare sull'etichetta tkinter
self.iterableCycle = cycle((PhotoImage(file=slide), slide)
                            per la diapositiva in listOfSlides)

# crea il widget tkinter Label che può visualizzare le immagini
self.slidesLabel = Etichetta(self)

# crea il widget Cornice
self.slidesLabel.pack()

def slidesCallback(self):
    # ottieni la diapositiva successiva dal ciclo iterabile
    currentInstance, nameOfSlide = next(self.iterableCycle)

    # assegna la diapositiva successiva al widget Etichetta
    self.slidesLabel.config(image=currentInstance)

    # aggiorna il titolo della finestra con la diapositiva corrente
    self.title(nameOfSlide)

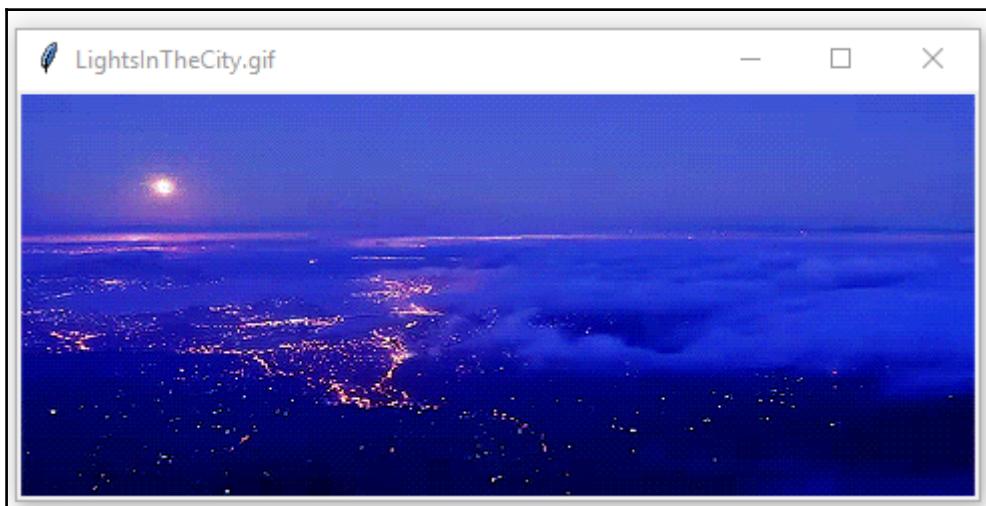
    # ripeti ricorsivamente lo spettacolo
    self.after(self.showTime, self.slidesCallback)

# =====
# Avvia GUI
# =====
vincere = SlideShow()
win.after(0, win.slidesCallback())
win.mainloop()
```

SlideShow.py



Ecco un altro momento nella presentazione in divenire:



Mentre lo scorimento delle diapositive è davvero impressionante, le capacità integrate di puro Python tkinter Le GUI non supportano il molto popolare .jpg format, quindi dobbiamo raggiungere un'altra libreria Python. Per utilizzare Cuscino, dobbiamo prima installarlo usando il pip comando.

Un'installazione riuscita si presenta come segue:

```
Command Prompt
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\Burkh>pip install pillow
Collecting pillow
  Downloading Pillow-4.0.0-cp36-cp36m-win_amd64.whl (1.5MB)
    100% |██████████| 1.5MB 506kB/s
Collecting olefile (from pillow)
  Downloading olefile-0.44.zip (74kB)
    100% |██████████| 81kB 1.6MB/s
Installing collected packages: olefile, pillow
  Running setup.py install for olefile ... done
Successfully installed olefile-0.44 pillow-4.0.0

C:\Users\Burkh>
```

Cuscino supporta.jpg formati e, per utilizzarlo, dobbiamo modificare leggermente la nostra sintassi.

Senza usare Cuscino, cercando di visualizzare un.jpg l'immagine genera il seguente errore:

SlideShow_try.jpg.py

```
# try: .jpeg
listOfSlides = [slide for slide in listdir() if slide.endswith('gif') or slide.endswith('jpg')]

Console Bookmarks
C:\Eclipse_NEON_workspace\2nd Edition Python GUI Programming Cookbook\Ch10_Code\SlideShow_try.jpg.py
File "C:\Python36\lib\tkinter\__init__.py", line 3495, in __init__
    self.tk.call(('image', 'create', imgtype, name,) + options)
_tkinter.TclError: couldn't recognize data in image file "rivers expedition day.jpg"
```

Usando Cuscino, possiamo visualizzare entrambi.gif e.jpg File. Dopo una corretta installazione del Cuscino pacchetto, dobbiamo solo apportare alcune modifiche:

```
④ # using Pillow instead of PIL (2.7) for Python 3.6
# Installation is: >pip install Pillow
from PIL import ImageTk

④ class SlideShow(Tk):
    # inherit GUI framework extending tkinter
④     def __init__(self, msShowTimeBetweenSlides=1500):
        # initialize tkinter super class
        Tk.__init__(self)

        # time each slide will be shown
        self.showTime = msShowTimeBetweenSlides

④     # look for images in current working directory where this module lives
        listOfSlides = [slide for slide in listdir() if slide.endswith('gif')]
        listOfSlides = [slide for slide in listdir() if slide.endswith('gif') or slide.endswith('jpg')]

④     # endlessly read in the slides so we can show them on the tkinter Label
        self.iterableCycle = cycle((PhotoImage(file=slide), slide) for slide in listOfSlides)
        self.iterableCycle = cycle((ImageTk.PhotoImage(file=slide), slide) for slide in listOfSlides)
```

Cerchiamo nella nostra cartella .jpg estensioni e, invece di utilizzare il FotoImmagine() classe, ora usiamo il ImmagineTk.FotoImmagine() classe. Questo ci permette di visualizzare immagini con diverse estensioni:

SlideShow_Pillow.py





Come funziona...

Python è uno strumento meraviglioso e, in questa ricetta, abbiamo esplorato diversi modi per usarlo ed estenderlo.

1 1

Migliori pratiche

In questo capitolo, esploreremo le migliori pratiche relative alla nostra GUI Python. Tratteremo le seguenti ricette:

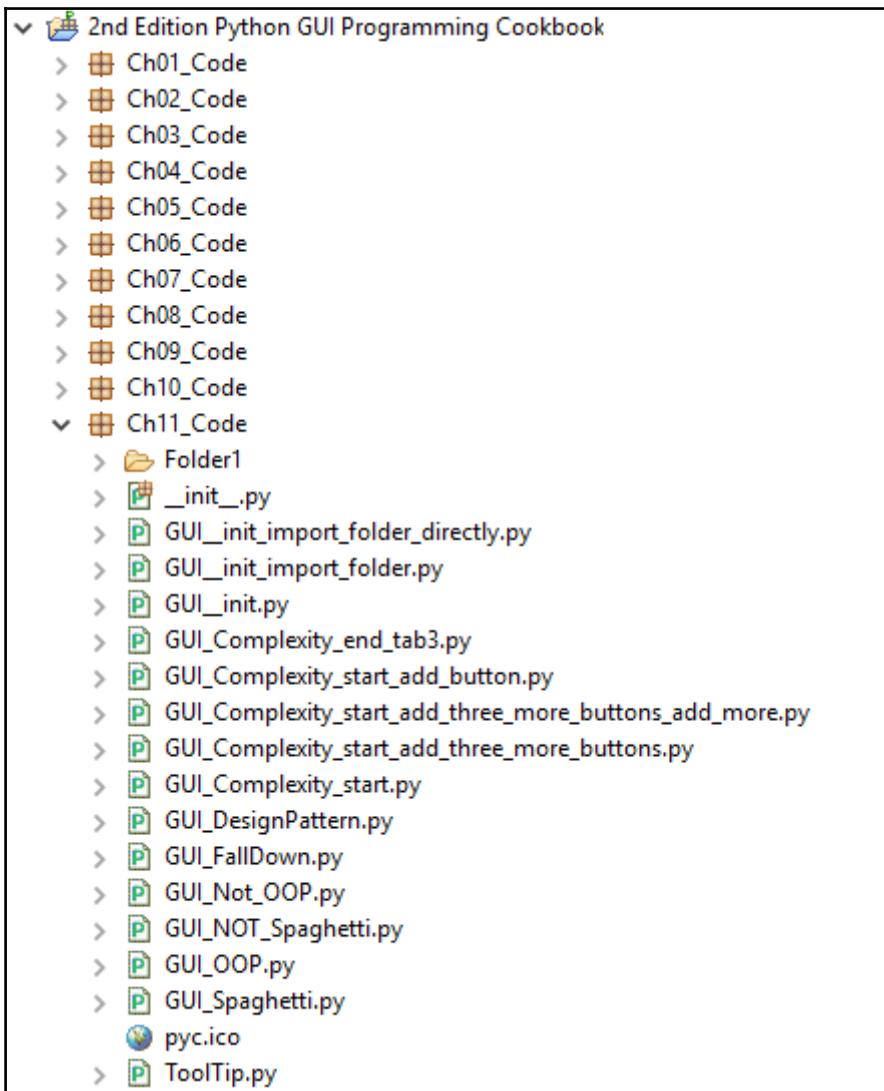
- Evitare il codice spaghetti Utilizzando `_dentro_`
- per connettere i moduli Mescolare la codifica
- fall-down e OOP Usare una convenzione di
- denominazione in codice Quando non usare
- OOP
- Come utilizzare con successo i modelli di
- progettazione Evitare la complessità
- Progettazione della GUI utilizzando più notebook

introduzione

In questo capitolo, esploreremo diverse best practice che possono aiutarci a costruire la nostra GUI in modo efficiente e mantenerla sia gestibile che estendibile.

Queste migliori pratiche ci aiuteranno anche a eseguire il debug della nostra GUI per ottenerla proprio come vogliamo che sia.

Ecco la panoramica dei moduli Python per questo capitolo:



Evitare il codice spaghetti

In questa ricetta, esploreremo un modo tipico per creare codice spaghetti e poi vedremo un modo molto migliore per evitare tale codice.



Il codice spaghetti è un codice in cui sono intrecciate molte funzionalità.

Prepararsi

Creeremo una nuova, semplice GUI, scritta in Python usando il Python integrato tkinker biblioteca.

Come farlo...

Dopo aver cercato online e letto la documentazione, potremmo iniziare scrivendo il seguente codice per creare la nostra GUI:

GUI_Spaghetti.py

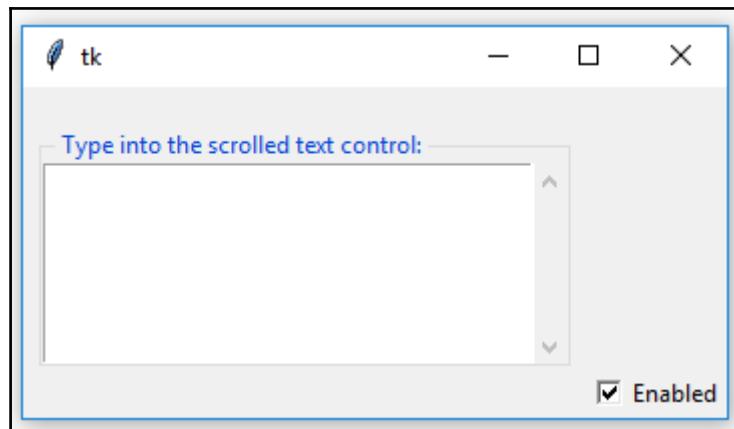
```
# Codice spaghetti #####
def PRINTME(me):print(me)
import tkinter
x=y=z=1
STAMPA(z)
da tkinter import *
scrollW=30;scrolH=6
win=tkinter.Tk()
if x:chVarUn=tkinter.IntVar() from
tkinter import ttk
NOI='NOI'
import tkinter.scrolledtext outputFrame=tkinter.ttk.LabelFrame(win,text=' Digita nel testo
scorrevole
controllo: ')
scr=tkinter.scrolledtext.ScrolledText(outputFrame,width=scrollW,height=scrol
H,wrap=tkinter.WORD)
e='E'
scr.grid(column=1,row=1,sticky=WE)
outputFrame.grid(column=0,row=2,sticky=e,padx=8)
IFrame=Nessuno
if y:chck2=tkinter.Checkbutton(IFrame,text="Enabled",variable=chVarUn) wE='WE'

se y==x:PRINTME(x)
IFrame=tkinter.ttk.LabelFrame(win,text="Spaghetti")
chck2.grid(column=1,row=4,sticky=tkinter.W,columnspan=3) PRINTME(z)

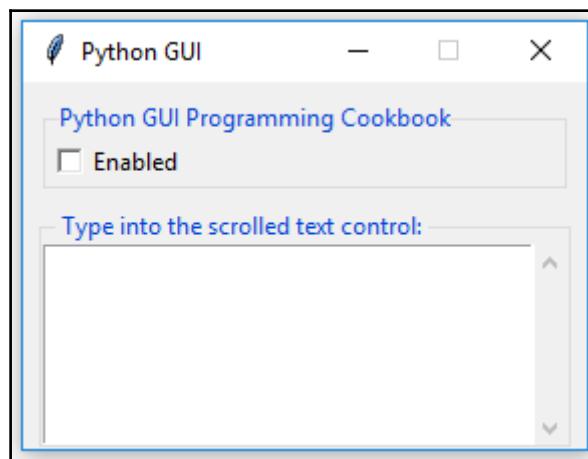
IFrame.grid(column=0,row=0,sticky=wE,padx=10,pady=10)
```

```
chck2.select()
prova: win.mainloop()
eccetto: PRINTME(x)
chck2.deselect()
se y==x:PRINTME(x)
# Fine Pasta #####
```

L'esecuzione del codice precedente risulta nella seguente GUI:



Questa non è proprio la GUI che intendevamo. Volevamo che assomigliasse a qualcosa di più simile a questo:



Mentre il codice spaghetti ha creato una GUI, il codice è molto difficile da eseguire il debug perché c'è così tanta confusione nel codice.

Quello che segue è il codice che produce la GUI desiderata:

GUI_NOT_Spaghetti.py

```
# =====
# importazioni
# =====
importa tkinter come tk da
tkinter importa ttk
da tkinter import scrolledtext

# =====
# Crea istanza
# =====
vincere = tk.Tk()

# =====
# Aggiungi un titolo
# =====
win.title("Python GUI")

# =====
# Disabilita il ridimensionamento della GUI
# =====
win.resizable(0,0)

# =====
# Aggiunta di un LabelFrame, una casella di testo (voce) e una casella combinata
# =====
lFrame = ttk.LabelFrame(win, text="Python GUI Programming Cookbook") lFrame.grid(column=0,
row=0, sticky='WE', padx=10 , paga=10)

# =====
# Utilizzo di un controllo di testo a scorrimento
# =====
outputFrame = ttk.LabelFrame(win, text=' Digita nel testo scorrevole
controllo: ')
outputFrame.grid(column=0, row=2, sticky='E', padx=8) scrollW = 30

scrollH = 6
scr = scrolledtext.ScrolledText(outputFrame, larghezza=scrollW,
altezza=scrollH, wrap=tk.WORD)
scr.grid(colonna=1, riga=0, sticky='NOI')

# =====
# Creazione di un pulsante di spunta
# =====
chVarUn = tk.IntVar()
```

```
check2 = tk.Checkbutton(lFrame, text="Enabled", variabile=chVarUn) check2.deselect()
check2.grid(column=1, row=4, sticky=tk.W, columnspan=3)

# =====
# Avvia GUI
# =====
win.mainloop()
```

Come funziona...

In questa ricetta, abbiamo confrontato il codice degli spaghetti con un buon codice. Un buon codice ha molti vantaggi rispetto al codice spaghetti.

Ha sezioni chiaramente commentate.

Quello che segue è un esempio di codice spaghetti:

```
def PRINTME(me):print(me)
import tkinter
x=y=z=1
STAMPA(z)
dall'importazione di tkinter *
```

Quello che segue è un esempio di buon codice:

```
# =====
# importazioni
# =====
importa tkinter come tk da
tkinter importa ttk
```

Ha un flusso naturale che segue il modo in cui i widget vengono disposti nella forma principale della GUI.

Nel codice spaghetti, il fondo EtichettaFrame viene creato prima della cima EtichettaFrame, ed è mescolato con un importare dichiarazione e qualche creazione di widget.

Considera il seguente esempio di codice spaghetti:

```
import tkinter.scrolledtext outputFrame=tkinter.ttk.LabelFrame(win,text=' Digita nel testo
scorrevole
controllo: ')
scr=tkinter.scrolledtext.ScrolledText(outputFrame,width=scrolW,height=scrol
H,wrap=tkinter.WORD)
e='E'
scr.grid(colonna=1,riga=1,appiccicoso=NOI)
```

```
outputFrame.grid(column=0, row=2, sticky=e, padx=8)
lFrame=Nessuno
if y==chck2==tkinter.Checkbutton(lFrame, text="Enabled", variable=chVarUn) wE='WE'

se y==x:PRINTME(x)
lFrame=tkinter.ttk.LabelFrame(win, text="Spaghetti")
```

Considera il seguente esempio di buon codice:

```
# =====
# Aggiunta di un LabelFrame, una casella di testo (voce) e una casella combinata
# ===== lFrame =
ttk.LabelFrame(win, text="Python GUI Programming Cookbook") lFrame.grid(column=0,
row=0, sticky='WE', padx=10 , pady=10)

# =====
# Utilizzo di un controllo di testo a scorrimento
# ===== outputFrame =
ttk.LabelFrame(win, text=' Digita nel testo scorrevole
controllo: ')
outputFrame.grid(column=0, row=2, sticky='E', padx=8)
```

Non contiene assegnazioni di variabili non necessarie e nemmeno ha un Stampa funzione che non esegue il debug che ci si potrebbe aspettare durante la lettura del codice.

Ad esempio, considera il seguente codice spaghetti:

```
def PRINTME(me):print(me)
x=y=z=1
e='E'
NOI='NOI'
scr.grid(column=1, row=1, sticky=NOI)
wE='NOI'
se y==x:PRINTME(x)
lFrame.grid(column=0, row=0, sticky=wE, padx=10, pady=10) PRINTME(z)

prova: win.mainloop()
eccetto: PRINTME(x)
chck2.deselect()
se y==x:PRINTME(x)
```

Un buon codice non ha nessuna delle istanze menzionate per il codice spaghetti.

Il importare le istruzioni importano solo i moduli richiesti. Non sono ingombranti in tutto il codice. Non ci sono duplicati importare dichiarazioni. Non c'è importare * dichiarazione.

Ancora una volta, diamo un'occhiata a un'altra istanza di codice spaghetti:

```
importare tkinter
x=y=z=1
STAMPA(z)
da tkinter import *
scrollW=30;scrollH=6
win=tkinter.Tk()
if x:chVarUn=tkinter.IntVar() from
tkinter import ttk
NOI='NOI'
import tkinter.scrolledtext
```

Ora, dai un'occhiata al buon codice:

```
importa tkinter come tk da
tkinter importa ttk
da tkinter import scrolledtext
```

I nomi delle variabili scelte sono abbastanza significativi. Non ci sono inutiliSe affermazioni che usano il numero 1 invece di Vero.

Quanto segue mostra il codice degli spaghetti:

```
x=y=z=1
if x:chVarUn=tkinter.IntVar() wE='NOI'
```

Quanto segue mostra un buon codice:

```
# =====
# Utilizzo di un controllo di testo a scorrimento
# ===== outputFrame =
ttk.LabelFrame(win, text=' Digita nel testo scorrevole
controllo: ')
outputFrame.grid(column=0, row=2, sticky='E', padx=8) scrollW = 30

scrollH = 6
scr = scrolledtext.ScrolledText(outputFrame, larghezza=scrollW,
altezza=scrollH, wrap=tk.WORD)
scr.grid(colonna=1, riga=0, sticky='NOI')
```

Non abbiamo perso il titolo della finestra previsto e il nostro pulsante di spunta è finito nella posizione corretta. Abbiamo anche fatto ilEtichettaFrame che circonda il pulsante di spunta visibile.

Codice spaghetti: GUI_Spaghetti.py

Abbiamo perso entrambi il titolo della finestra e non abbiamo visualizzato la parte superiore EtichettaFrame. Il pulsante Verifica è finito nel posto sbagliato.

Buon codice: GUI_NOT_Spaghetti.py

```
# =====
# Crea istanza
# =====
vincere = tk.Tk()

# =====
# Aggiungi un titolo
# =====
win.title("Python GUI")

# =====
# Aggiunta di un LabelFrame, una casella di testo (voce) e una casella combinata
# =====
lFrame = ttk.LabelFrame(win, text="Python GUI Programming Cookbook") lFrame.grid(column=0,
row=0, sticky='WE', padx=10 , paga=10)

# =====
# Creazione di un pulsante di spunta
# =====
chVarUn = tk.IntVar()
check2 = tk.Checkbutton(lFrame, text="Enabled", variabile=chVarUn) check2.deselect()

check2.grid(column=1, row=4, sticky=tk.W, columnspan=3)

# =====
# Avvia GUI
# =====
win.mainloop()
```

Usare `_init_` per connettere i moduli

Quando creiamo un nuovo pacchetto Python utilizzando il plug-in PyDev per l'IDE Eclipse, viene creato automaticamente un `_init_.py` modulo. Possiamo anche crearlo manualmente, quando non utilizziamo Eclipse.



Il `_init_.py` module è solitamente vuoto e, quindi, ha una dimensione di 0 kilobyte.

Possiamo usare questo modulo solitamente vuoto per connettere diversi moduli Python inserendo il codice in esso. Questa ricetta ti mostrerà come farlo.

Prepararsi

Creeremo una nuova GUI simile a quella che abbiamo creato nella ricetta precedente, *Evitare il codice spaghetti*.

Come farlo...

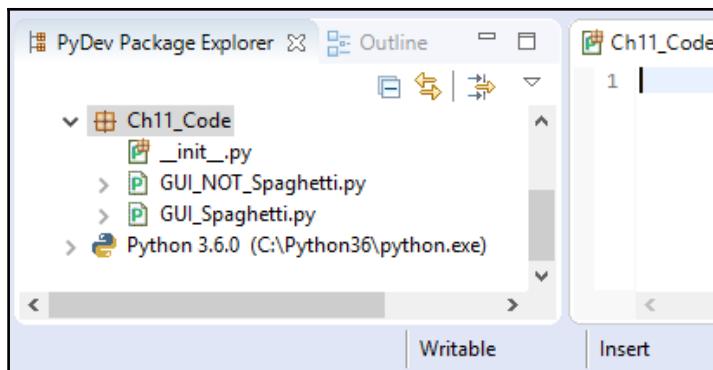
Man mano che il nostro progetto diventa sempre più grande, lo suddividiamo naturalmente in diversi moduli Python. A volte può essere un po' complicato trovare moduli che si trovano in diverse sottocartelle, sopra o sotto il codice che deve importarlo.

Un modo pratico per aggirare questa limitazione è usare `__init__.py` modulo.



In Eclipse, possiamo impostare il progetto interno di Eclipse PyDev **PYTHONPATH** in determinate cartelle e il nostro codice Python lo troverà. Ma al di fuori di Eclipse, ad esempio, quando si esegue da una finestra di comando, a volte c'è una mancata corrispondenza nel meccanismo di importazione del modulo Python e il codice non verrà eseguito.

Ecco uno screenshot del vuoto `__init__.py` modulo che non appare con il nome `__dentro__` ma con il nome del PyDev pacchetto a cui appartiene quando viene aperto nell'editor di codice Eclipse. Il 1 sul lato sinistro dell'editor di codice c'è il numero di riga e non il codice scritto in questo modulo. Non c'è assolutamente alcun codice in questo `_vuotoinit_.py` modulo:



Questo file è vuoto ma esiste:

Ch11_Code	
Name	Size
__init__.py	0 KB
GUI_NOT_Spaghetti.py	2 KB
GUI_Spaghetti.py	2 KB

Quando eseguiamo il seguente codice e facciamo clic su **clickami** pulsante, otteniamo il risultato mostrato post il codice. Questo è un normale modulo Python che non usa ancora __init__.py codice modulo:



Il __init__.py modulo non è lo stesso di __in se stesso):
metodo di una classe Python.

```
# GUI_init.py
# =====
# importazioni
# =====
importa tkinter come tk da
tkinter importa ttk

# =====
# Crea istanza
# =====
vincere = tk.Tk()

# =====
# Aggiungi un titolo
# =====
win.title("Python GUI")

# =====
# Aggiunta di un LabelFrame e di un pulsante
# =====
lFrame =
ttk.LabelFrame(win, text="Python GUI Programming Cookbook") lFrame.grid(column=0,
row=0, sticky='WE', padx=10 , paga=10)

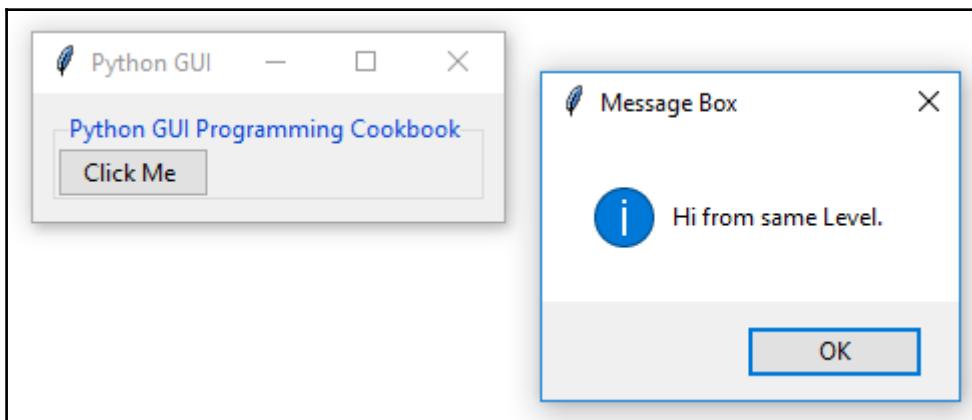
def clickMe():
    da tkinter import messagebox
```

```
messagebox.showinfo('Casella messaggi', 'Ciao dallo stesso livello.')

button = ttk.Button(lFrame, text="Click Me", command=clickMe)
button.grid(column=1, row=0, sticky=tk.S)

# =====
# Avvia GUI
# =====
win.mainloop()
```

Esecuzione del codice precedente denominato `GUI_init.py`, otteniamo il seguente output:



Nel codice precedente, abbiamo creato la seguente funzione, che importa la finestra di messaggio di Python e poi la usa per visualizzare la finestra di dialogo della finestra di messaggio:

```
def clickMe():
    from tkinter import messagebox
    messagebox.showinfo('Message Box', 'Ciao
dallo stesso livello.')
```

Quando spostiamo il `cliccami()` codice della casella di messaggio in una directory annidata e provare a importare nel nostro modulo `GUI`, potremmo imbatterci in alcune sfide.

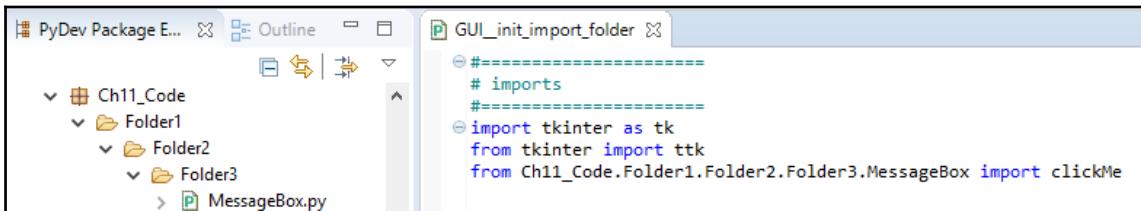
Abbiamo creato tre sottocartelle al di sotto dove risiede il nostro modulo Python. Abbiamo quindi posizionato il `cliccami()` codice della casella di messaggio in un nuovo modulo Python, che abbiamo chiamato `MessageBox.py`. Questo modulo vive in `Cartella3`, tre livelli al di sotto del punto in cui risiede il nostro modulo Python.

Dobbiamo importare questo `MessageBox.py` modulo per utilizzare il `cliccami()` funzione che questo modulo contiene.

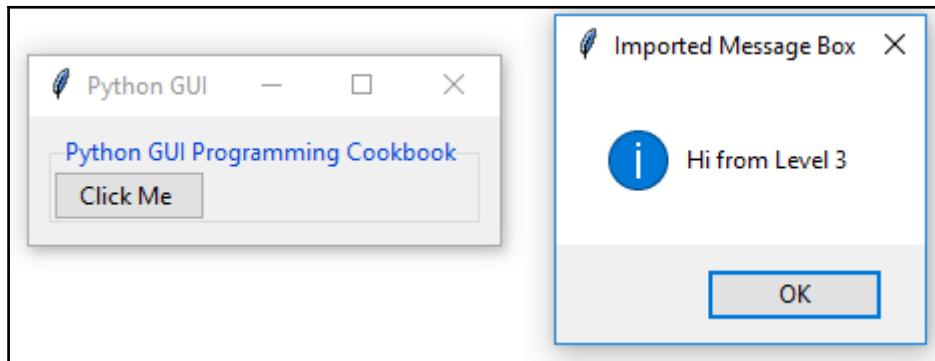
Usiamo la sintassi di importazione relativa di Python:

```
da Ch11_Code.Folder1.Folder2.Folder3.MessageBox importa clickme
```

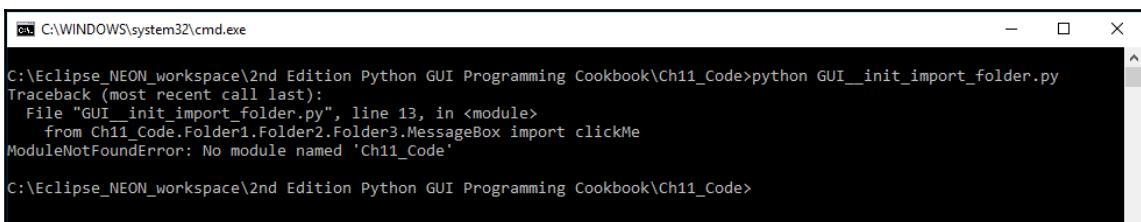
L'istruzione di importazione e la struttura delle cartelle possono essere visualizzate nello screenshot seguente:



Abbiamo cancellato il locale cliccami() funzione e ora il nostro callback dovrebbe utilizzare l'importato cliccami() funzione. Funziona da Eclipse. Esecuzione del file di codice GUI_init_import_folder.py con le modifiche precedenti dà il seguente risultato:



Quando lo eseguiamo da un prompt dei comandi, riceviamo un errore:



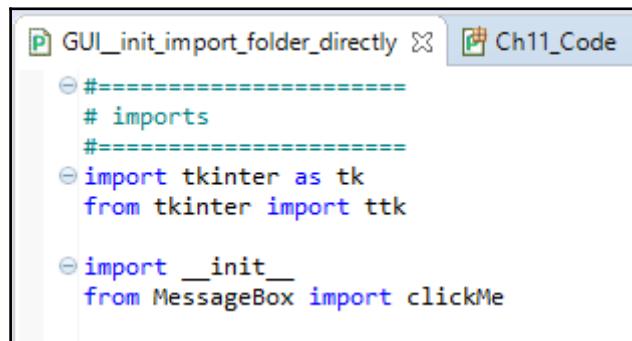
Per risolvere questo errore, possiamo inizializzare il nostro percorso di ricerca Python all'interno di `__init__.py` modulo aggiungendo il seguente codice al `__init__.py` modulo:

```
print('ciao from GUI init\n') from sys
import pathsys
da pprint importa pprint
# =====
# Configurazione richiesta per PYTONPATH per trovare tutte le cartelle dei pacchetti
# ===== dal sito
import addsitesdir
da os import.getcwd, chdir, pardir mentre True:

curFull = getcwd()
curDir = curFull.split('\\')[-1] if 'Ch11_Code'
== curDir:
    addsitesdir(curFull)
    addsitesdir(curFull + 'Cartella1\Cartella2\Cartella3') break

    chdir(pardir)
pprint(percorso)
```

Nel modulo in cui importiamo il cliccammi funzione, non è più necessario specificare il percorso completo della cartella. Possiamo importare direttamente il modulo e la sua funzione:



```
GUI_init_import_folder_directly.py Ch11_Code
=====
# imports
=====
import tkinter as tk
from tkinter import ttk

import __init__
from MessageBox import clickMe
```



Dobbiamo esplicitamente importa `__init__` affinché questo codice funzioni.

Quando ora eseguiamo il nostro codice GUI, otteniamo le stesse finestre di prima, ma abbiamo rimosso la nostra dipendenza da Eclipse PYTHONPATH variabile.

Ora possiamo eseguire con successo lo stesso codice all'esterno di Eclipse PyDev collegare:

The screenshot shows a terminal window on the left and two windows on the right. The terminal window displays the command 'python GUI_init_import_folder_directly.py' and its output, which lists several paths starting with 'C:\'. The two windows are titled 'Python GUI' and 'Imported Message Box'. The 'Python GUI' window contains the text 'Python GUI Programming Cookbook' and a button labeled 'Click Me'. The 'Imported Message Box' window contains the text 'Hi from Level 3' and a button labeled 'OK'.

```
python GUI_init_import_folder_directly.py
C:\Eclipse_NEON_workspace\2nd Edition Python GUI Programming Cookbook\Ch11_Code>python GUI_init_import_folder_directly.py
hi from GUI init
['C:\\\\Eclipse_NEON_workspace\\\\2nd Edition Python GUI Programming ',
 'Cookbook\\\\Ch11_Code',
 'C:\\\\Eclipse_NEON_workspace\\\\2nd Edition Python GUI Programming ',
 'Cookbook\\\\Ch11_Code\\\\xC:\\\\Eclipse_NEON_workspace\\\\2nd Edition Python GUI ',
 'Programming Cookbook',
 'C:\\\\Python36\\\\python36.zip',
 'C:\\\\Python36\\\\DLLs',
 'C:\\\\Python36\\\\lib',
 'C:\\\\Python36',
 'C:\\\\Python36\\\\lib\\\\site-packages',
 'C:\\\\Python36\\\\lib\\\\site-packages\\\\win32',
 'C:\\\\Python36\\\\lib\\\\site-packages\\\\win32\\\\lib',
 'C:\\\\Python36\\\\lib\\\\site-packages\\\\Pythonwin',
 'C:\\\\Eclipse_NEON_workspace\\\\2nd Edition Python GUI Programming ',
 'Cookbook\\\\Ch11_Code\\\\Folder1\\\\Folder2\\\\Folder3']
```

Il nostro codice è diventato più Pythonic.



Come funziona...

In questa ricetta, abbiamo importato una funzione da un modulo annidato in una struttura di cartelle.

Abbiamo visto che possiamo usare `__init__.py` modulo per aggiungere la struttura delle cartelle al percorso di ricerca di Python. Lo abbiamo fatto aggiungendo prima il codice e poi importando esplicitamente il explicitly `__init__.py` modulo.

In un ciclo, cerchiamo la struttura della directory finché non troviamo la cartella principale che stiamo cercando.

Quindi otteniamo il percorso completo di questa cartella e aggiungiamo le tre cartelle. Non stiamo codificando l'intero percorso di ricerca. Aggiungiamo solo le cartelle note ovunque si trovi il nostro modulo.

Usando il codice che abbiamo aggiunto a `__init__.py` Il modulo ci ha permesso di eseguire il codice con successo sia dall'interno di Eclipse che da un prompt dei comandi.

Usare Python puro è solitamente il modo migliore per andare.



Miscelazione fall-down e codifica OOP

Python è un linguaggio di programmazione orientato agli oggetti, ma non sempre ha senso usare OOP. Per semplici attività di scripting, lo stile di codifica a cascata legacy è ancora appropriato.

In questa ricetta, creeremo una nuova GUI che mescoli sia lo stile di codifica fall-down con lo stile di codifica OOP più moderno.

Creeremo una classe in stile OOP che mostrerà un tooltip quando passeremo il mouse su un widget in una GUI Python che creeremo usando lo stile a cascata.



Gli stili di codifica Fall-down e Waterfall sono gli stessi. Significa che dobbiamo posizionare fisicamente il codice sopra il codice prima di poterlo chiamare dal codice sottostante. In questo paradigma, il codice cade letteralmente dalla parte superiore del nostro programma alla parte inferiore del nostro programma quando eseguiamo il codice.

Prepararsi

In questa ricetta, creeremo una GUI usando tkinter, che è simile alla GUI che abbiamo creato nel primo capitolo di questo libro.

Come farlo...

In Python, possiamo associare funzioni alle classi trasformandole in metodi usando il se stesso convenzione di denominazione. Questa è una capacità davvero meravigliosa di Python e ci consente di creare sistemi di grandi dimensioni comprensibili e mantenibili.

A volte, quando scriviamo solo brevi script, l'OOP non ha senso perché ci troviamo ad anteporre molte variabili con il se stesso convenzione di denominazione e il codice diventa inutilmente grande quando non è necessario.

Creiamo prima una GUI Python usando tkinter e codificalo nello stile a cascata. Il

seguente codice `GUI_FallDown.py` crea la GUI:

```
# =====
# importazioni
# =====
import tkinter as tk
tkinter import ttk
from tkinter import messagebox
```

```
# =====
# Crea istanza
# =====
vincere = tk.Tk()

# =====
# Aggiungi un titolo
# =====
win.title("Python GUI")

# =====
# Disabilita il ridimensionamento della GUI
# =====
win.resizable(0,0)

# =====
# Aggiunta di un LabelFrame, una casella di testo (voce) e una casella combinata
# =====
lFrame = ttk.LabelFrame(win, text="Python GUI Programming Cookbook") lFrame.grid(column=0, row=0, sticky='WE', padx=10 , paga=10)

# =====
# Etichette
# =====
ttk.Label(lFrame, text="Inserisci un nome:").grid(column=0, row=0) ttk.Label(lFrame, text="Scegli un numero :").grid(column=1, row=0, sticky=tk.W)

# =====
# Comando clic sui pulsanti
# =====
def clickMe(nome, numero):
    messagebox.showinfo('Casella messaggio informativo', 'Ciao '+nome+
                       ', il tuo numero è: ' + numero)

# =====
# Creazione di più controlli in un ciclo
# =====
nomi = ['nome0', 'nome1', 'nome2']
nameEntries = ['nameEntry0', 'nameEntry1', 'nameEntry2']

numeri      = ['numero0', 'numero1', 'numero2']
numberEntry = ['numberEntry0', 'numberEntry1', 'numberEntry2']

pulsanti = []

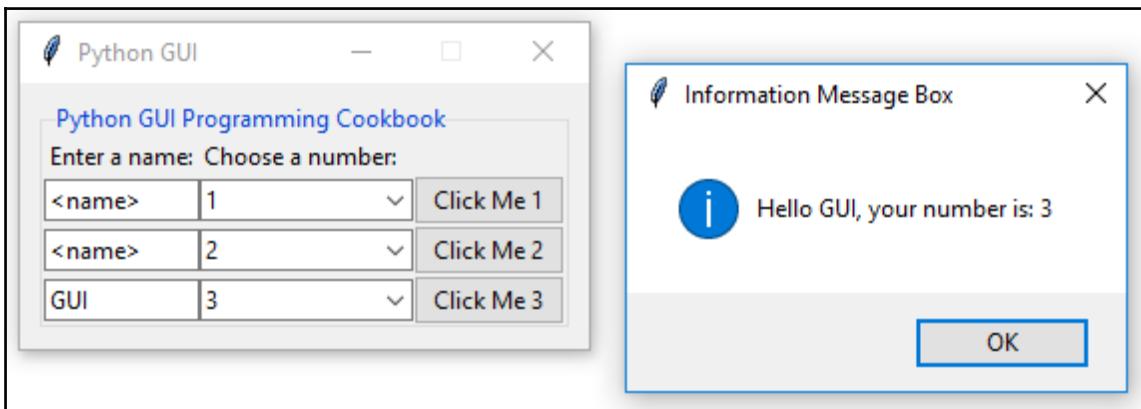
per idx nell'intervallo (3):
    nomi[idx] = tk.StringVar()
```

```
nameEntries[idx] = ttk.Entry(lFrame, width=12, textvariable=names[idx])
nameEntries[idx].grid(column=0, row=idx+1)
nameEntries[idx].delete(0, tk.END)
nameEntries[idx].insert(0, '<nome>')

numeri[idx] = tk.StringVar()
numberEntries[idx] = ttk.Combobox(lFrame, larghezza=14,
                                 textvariable=numeri[idx])
numberEntries[idx]['values'] = (1+idx, 2+idx, 4+idx, 42+idx, 100+idx)
numberEntries[idx].grid(column=1, row=idx+1)
numeroVoci[idx].corrente(0)

button = ttk.Button(lFrame, text="Fai clic su di me"+str(idx+1),
                    comando=lambda idx=idx: clickMe(nomi[idx].get(),
                                                 numeri[idx].get()))
button.grid(column=2, row=idx+1, sticky=tk.W)
button.append(button)
# =====
# Avvia GUI
# =====
win.mainloop()
```

Quando eseguiamo il codice, otteniamo la GUI e appare così:



Possiamo migliorare la nostra GUI Python aggiungendo suggerimenti. Il modo migliore per farlo è isolare il codice che crea la funzionalità di tooltip dalla nostra GUI.

Lo facciamo creando una classe separata, che ha la funzionalità di tooltip, e poi creiamo un'istanza di questa classe nello stesso modulo Python che crea la nostra GUI.

Usando Python, non è necessario posizionare il nostro Descrizione comando classe in un modulo separato. Possiamo posizionarlo appena sopra il codice procedurale e quindi chiamarlo da sotto il codice della classe.

Il codice sarà il seguente:

```
# =====
# importazioni
# =====
importa tkinter come tk da
tkinter importa ttk
da tkinter import messagebox

# -----
classe ToolTip (oggetto):
    def __init__(self, widget):
        self.widget = widget
        self.tipwindow = Nessuno
        self.id = Nessuno
        auto.x = auto.y = 0

    # -----
    def createToolTip(widget, testo):
        toolTip = ToolTip(widget)
        def enter(event): toolTip.showtip(text) def
        leave(event): toolTip.hidetip() widget.bind('<Enter>',
        enter)
        widget.bind('<Lascia>', lascia)

    # -----
    # più in basso nel modulo chiamiamo la funzione createToolTip
    # -----


per idx nell'intervallo (3):
    nomi[idx] = tk.StringVar()
    nameEntries[idx] = ttk.Entry(lFrame, width=12, textvariable=names[idx])
    nameEntries[idx].grid(column=0, row=idx+1)
    nameEntries[idx].delete(0, tk.END)
    nameEntries[idx].insert(0, '<nome>')

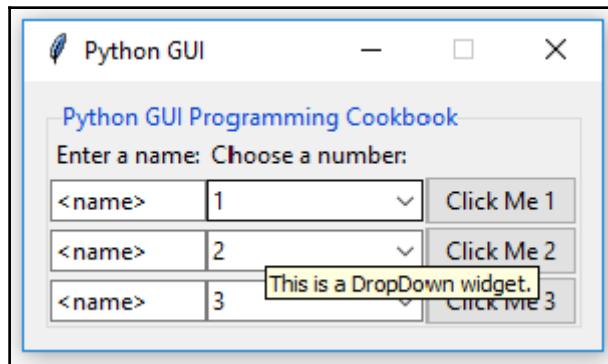
    numeri[idx] = tk.StringVar()
    numberEntries[idx] = ttk.Combobox(lFrame, larghezza=14,
                                    textvariable=numeri[idx])
    numberEntries[idx]['values'] = (1+idx, 2+idx, 4+idx, 42+idx, 100+idx)
    numberEntries[idx].grid(column=1, row=idx+1)
    numeroVoci[idx].corrente(0)

    button = ttk.Button(lFrame, text="Fai clic su di me"+str(idx+1),
                        comando=lambda idx=idx: clickMe(nomi[idx].get(),
```

```
        numeri[idx].get()))
button.grid(column=2, row=idx+1, sticky=tk.W)
button.append(button)

# -----
# Aggiungi suggerimenti a più widget
createToolTip(nameEntries[idx], 'Questo è un widget Entry.')
createToolTip(numberEntries[idx], 'Questo è un widget DropDown.')
createToolTip(buttons[idx], 'Questo è un widget Button.')
# -----
```

L'esecuzione del codice crea suggerimenti per i nostri widget quando passiamo il mouse su di essi:



Come funziona...

In questa ricetta, abbiamo creato una GUI Python in modo procedurale e successivamente abbiamo aggiunto una classe all'inizio del modulo.

Possiamo facilmente combinare e abbinare sia la programmazione procedurale che quella OOP nello stesso modulo Python.

Utilizzo di una convenzione di denominazione in codice

Le prime ricette del *Prima edizione* di questo libro non utilizzava una convenzione di denominazione in codice strutturata. Questa ricetta ti mostrerà il valore di aderire a uno schema di denominazione del codice perché ci aiuta a trovare il codice che vogliamo estendere, oltre a ricordarci il design del nostro programma.

Prepararsi

In questa ricetta, esamineremo i nomi dei moduli Python dal primo capitolo del *Prima edizione* di questo libro e confrontarli con migliori convenzioni di denominazione.

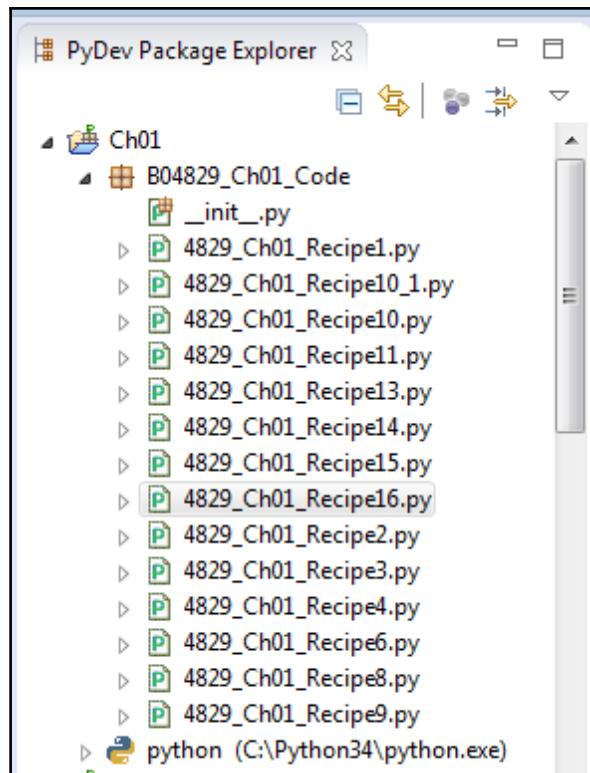
Come farlo...

Nel primo capitolo di questo libro abbiamo creato la nostra prima GUI Python. Abbiamo migliorato la nostra GUI incrementando i diversi nomi dei moduli di codice tramite numeri sequenziali.

Le schermate seguenti sono tratte dalla prima edizione di questo libro.



Sembrava così:



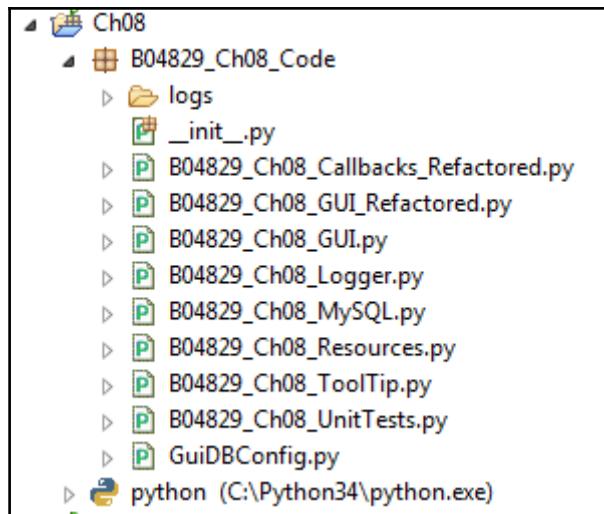
Sebbene questo sia un modo tipico di codificare, non fornisce molto significato. Quando scriviamo il nostro codice Python durante lo sviluppo, è molto facile incrementare i numeri.

Più tardi, tornando a questo codice, non abbiamo molta idea di quale modulo Python fornisca quale funzionalità e, a volte, i nostri ultimi moduli incrementati non sono buoni come le versioni precedenti.



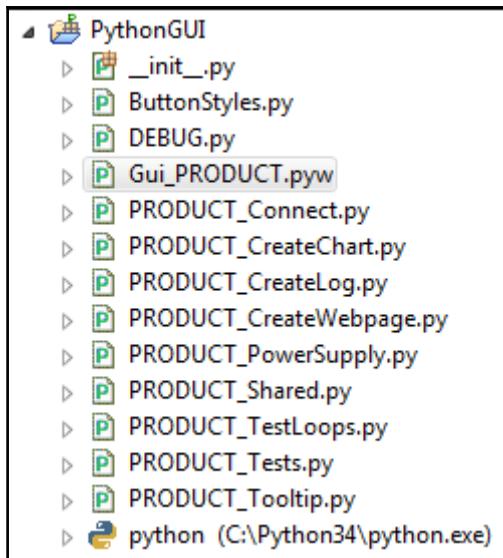
Una chiara convenzione di denominazione aiuta.

Possiamo confrontare i nomi dei moduli da Capitolo 1, *Creazione del modulo GUI e aggiunta di widget*, ai nomi da capitolo 8, *Internazionalizzazione e Sperimentazione*, che sono molto più significativi:



Sebbene non perfetti, i nomi scelti per i diversi moduli Python indicano qual è la responsabilità di ciascun modulo. Quando vogliamo aggiungere più unit test, è chiaro in quale modulo risiedono.

La seguente convenzione di denominazione del codice è un altro esempio di come utilizzare una convenzione di denominazione del codice per creare una GUI in Python:



Sostituisci la parola PRODOTTO con il prodotto su cui stai lavorando.



L'intera applicazione è una GUI. Tutte le parti sono collegate. Il DEBUG.py modulo viene utilizzato solo per il debug del nostro codice. La funzione principale per richiamare la GUI ha il nome invertito rispetto a tutti gli altri moduli. Si inizia con Gui e termina con a.pyw estensione.

È l'unico modulo Python che ha questo nome di estensione.

Da questa convenzione di denominazione, se hai abbastanza familiarità con Python, sarà ovvio che, per eseguire questa GUI, devi fare doppio clic su Gui_PRODUCT.pyw modulo.

Tutti gli altri moduli Python contengono funzionalità da fornire alla GUI ed eseguire la logica di business sottostante per soddisfare lo scopo a cui si rivolge questa GUI.

Come funziona...

Le convenzioni di denominazione per i moduli di codice Python sono di grande aiuto per mantenerci efficienti e aiutarci a ricordare il nostro design originale. Quando abbiamo bisogno di eseguire il debug e correggere un difetto o aggiungere una nuova funzionalità, sono le prime risorse da esaminare.



L'incremento dei nomi dei moduli in base ai numeri non è molto significativo e alla fine fa perdere tempo allo sviluppo.

D'altra parte, nominare le variabili Python è più una forma libera. Python deduce i tipi, quindi non dobbiamo specificare che una variabile sarà di tipo <lista> (potrebbe non esserlo o, più avanti nel codice, potrebbe diventare di un tipo diverso).

Una buona idea per nominare le variabili è renderle descrittive ed è anche una buona idea non abbreviare troppo.

Se vogliamo far notare che una certa variabile è progettata per essere del <lista> digita, allora è molto più intuitivo usare la parola intera elenco invece di lst.

È simile per numero invece di numero

Sebbene sia una buona idea avere nomi molto descrittivi per le variabili, a volte ciò può diventare troppo lungo. Nel linguaggio Objective-C di Apple, alcuni nomi di variabili e funzioni sono estremi: questo è un metodo che fa questo e quello e anche quello se passi in N Interi:1:2:3



Usare il buon senso quando si nominano variabili, metodi e funzioni.

Quando non usare OOP

Python è dotato di funzionalità di programmazione orientate agli oggetti, ma allo stesso tempo possiamo scrivere script che non necessitano di utilizzare l'OOP. Per alcune attività, l'OOP non ha senso.

Questa ricetta ci mostrerà quando non usare l'OOP.

Prepararsi

In questa ricetta creeremo una GUI Python simile alle ricette precedenti. Confronteremo il codice OOP con il modo alternativo di programmazione non OOP.

Come farlo...

Creiamo prima una nuova GUI utilizzando la metodologia OOP. Il codice seguente creerà la GUI visualizzata, succedendo al codice:

GUI_Not_OOP.py

```
importa tkinter come tk
tkinter importa ttk
from tkinter import scrolledtext
from tkinter import Menu

classe OOP():
    def __init__(self):
        self.win = tk.Tk()
        self.win.title ("GUI Python")
        self.createWidgets()

    def createWidgets(self):
        tabControl = ttk.Notebook(self.win) tab1 =
            ttk.Frame(tabControl) tabControl.add(tab1,
            text='Tab 1') tabControl.pack(expand=1, fill="both")

        self.monty = ttk.LabelFrame(tab1, text=' Mighty Python ')
        self.monty.grid(column=0, row=0, padx=8, pady=4)

        ttk.Label(self.monty, text="Inserisci un nome:").grid(column=0,
            riga=0, appiccicoso='W')
        self.name = tk.StringVar()
        nomeEntrato = ttk.Entry(self.monty, larghezza=12,
            textvariable=self.name)
        nameEntered.grid(column=0, row=1, sticky='W')

        self.action = ttk.Button(self.monty, text="Click Me!")
        self.action.grid(column=2, row=1)

        ttk.Label(self.monty, text="Scegli un numero:")
            . griglia(colonna=1, riga=0)
        numero = tk.StringVar()
        numeroScelto = ttk.Combobox(self.monty, larghezza=12,
            variabile di testo=numero)
        numberChosen['values'] = (42)
        numberChosen.grid(column=1, row=1)
        numberChosen.current(0)

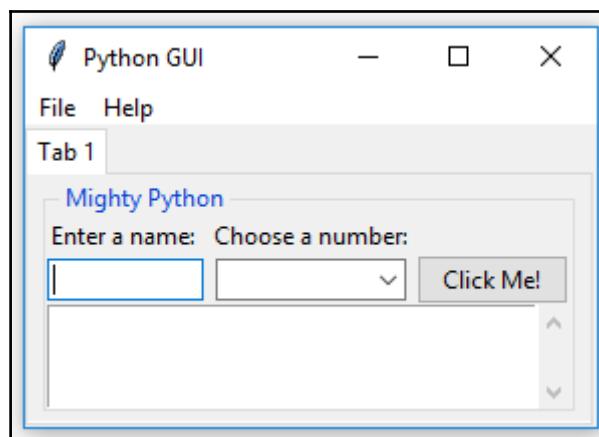
        scrollW = 30; scrollH = 3
        self.scr = scrolledtext.ScrolledText(self.monty, width=scrollW,
            altezza=scrollH, wrap=tk.WORD)
```

```
self.scr.grid(column=0, row=3, sticky='WE', columnspan=3)

menuBar = Menu(tab1)
self.win.config(menu=menuBar) fileMenu = Menu(menuBar,
tearoff=0) menuBar.add_cascade(label="File", menu=fileMenu)
helpMenu = Menu(menuBar, tearoff=0)

menuBar.add_cascade(label="Aiuto", menu=helpMenu)

nomeEntrato.focus()
# =====
oop = OOP()
oop.win.mainloop()
```



Possiamo ottenere la stessa GUI senza utilizzare un approccio OOP ristrutturando leggermente il nostro codice. Per prima cosa, rimuoviamo il OOP classe e il suo _dentro_ metodo.

Successivamente, spostiamo tutti i metodi a sinistra e rimuoviamo il se stesso riferimento di classe, che li trasforma in funzioni non associate.

Rimuoviamo anche qualsiasi altro se stesso riferimenti che il nostro codice precedente aveva. Quindi, spostiamo il creare widget chiamata di funzione sotto il punto della dichiarazione della funzione. Lo posizioniamo appena sopra il ciclo principale chiamata.

Alla fine, otteniamo la stessa GUI ma senza usare l'OOP. Il codice rifattorizzato è mostrato come segue:

```
importa tkinter come tk da
tkinter importa ttk
da tkinter import scrolledtext
```

dal menu di importazione di tkinter

```
def createWidgets():
    tabControl = ttk.Notebook(win) tab1 =
        ttk.Frame(tabControl) tabControl.add(tab1,
        text='Tab 1') tabControl.pack(expand=1, fill="both")

    monty = ttk.LabelFrame(tab1, text=' Mighty Python ')
    monty.grid(column=0, row=0, padx=8, pady=4)

    ttk.Label(monty, text="Inserisci un nome:").grid(column=0, row=0,
                                                    appiccoso='W')
    nome = tk.StringVar()
    nameEntered = ttk.Entry(monty, width=12, textvariable=name)
    nameEntered.grid(column=0, row=1, sticky='W')

    action = ttk.Button(monty, text="Click Me!")
    action.grid(column=2, row=1)

    ttk.Label(monty, text="Scegli un numero:").grid(column=1, row=0) numero =
    tk.StringVar()
    numberChosen = ttk.Combobox(monty, width=12, textvariable=number)
    numberChosen['values'] = (42)
    numeroScelto.grid(colonna=1, riga=1)
    numeroScelto.corrente(0)

    scrollW = 30; scrollH = 3
    scr = scrolledtext.ScrolledText(monty, width=scrollW,
                                    altezza=scrollH, wrap=tk.WORD)
    scr.grid(column=0, row=3, sticky='NOI', columnspan=3)

    menuBar = Menu(tab1)
    win.config(menu=menuBar)
    fileMenu = Menu(menuBar, tearoff=0)
    menuBar.add_cascade(label="File", menu=fileMenu) helpMenu =
    Menu(menuBar, tearoff=0)
    menuBar.add_cascade(label="Aiuto", menu=helpMenu)

    nomeEntrato.focus()

# =====
vincere = tk.Tk()
win.title("GUI Python")
createWidgets()
win.mainloop()
```

Come funziona...

Python ci consente di utilizzare l'OOP quando ha senso. Altri linguaggi come Java e C# ci costringono a utilizzare sempre l'approccio OOP alla codifica. In questa ricetta, abbiamo esplorato una situazione in cui non aveva senso utilizzare l'OOP.



L'approccio OOP sarà più estensibile se la base di codice cresce, ma se è certo che è l'unico codice necessario, non è necessario passare attraverso l'OOP.

Come utilizzare con successo i modelli di progettazione

In questa ricetta, creeremo widget per la nostra GUI Python utilizzando il modello di progettazione di fabbrica. Nelle ricette precedenti, abbiamo creato i nostri widget manualmente uno alla volta o dinamicamente in loop. Utilizzando il modello di progettazione di fabbrica, utilizzeremo la fabbrica per creare i nostri widget.

Prepararsi

Creeremo una GUI Python che ha tre pulsanti, ognuno con stili diversi.

Come farlo...

Verso la parte superiore del nostro modulo GUI Python, appena sotto le istruzioni di importazione, creiamo diverse classi:

```
importa tkinter come tk da
tkinter importa ttk
from tkinter import scrolledtext from
tkinter import Menu

classe ButtonFactory():
    def createButton(self, type_):
        pulsante di ritornoTipi[tipos_]()

classe ButtonBase():
    rilievo = 'piatto'
    primo piano = 'bianco'
    def getButtonConfig(self):
        ritorno self.relief, self.foreground
```

```
classe ButtonRidge(ButtonBase):
    rilievo = 'cresta'
    primo piano = 'rosso'

classe ButtonSunken(ButtonBase):
    rilievo = 'affondato'
    primo piano = 'blu'

classe ButtonGroove(ButtonBase):
    rilievo = 'scanalatura'
    primo piano = 'verde'

buttonTypes = [ButtonRidge, ButtonSunken, ButtonGroove]

classe OOP():
    def __init__(self):
        self.win = tk.Tk()
        self.win.title ("GUI Python")
        self.createWidgets()
```

Creiamo una classe base da cui ereditano le nostre diverse classi di stile dei pulsanti e in cui ognuna di esse sovrascrive il sollevo e primo piano proprietà di configurazione. Tutte le sottoclassi ereditano il getButtonConfig metodo da questa classe base. Questo metodo restituisce una tupla.

Creiamo anche una classe di fabbrica di pulsanti e un elenco che contiene i nomi delle nostre sottoclassi di pulsanti. Diamo un nome alla listabuttonTipi, poiché la nostra fabbrica creerà diversi tipi di pulsanti.

Più in basso nel modulo, creiamo i widget dei pulsanti, usando lo stesso buttonTypes

elenco:

```
def createButtons(self):

    fabbrica = ButtonFactory()

    # Pulsante 1
    rel = factory.createButton(0).getButtonConfig()[0] fg =
    factory.createButton(0).getButtonConfig()[1] action = tk.Button(self.monty,
    text="Button "+str(0+ 1),
                           rilievo=rel, primo piano=fg)
    action.grid(colonna=0, riga=1)

    # Pulsante 2
    rel = factory.createButton(1).getButtonConfig()[0] fg =
    factory.createButton(1).getButtonConfig()[1] action = tk.Button(self.monty,
    text="Button "+str(1+ 1),
                           rilievo=rel, primo piano=fg)
```

```
action.grid(colonna=1, riga=1)

# Pulsante 3
rel = factory.createButton(2).getButtonConfig()[0] fg =
factory.createButton(2).getButtonConfig()[1] action = tk.Button(self.monty,
text="Button "+str(2+1),
relief=rel, primo piano=fg)
action.grid(colonna=2, riga=1)
```

Innanzitutto, creiamo un'istanza della fabbrica di pulsanti e quindi utilizziamo la nostra fabbrica per creare i nostri pulsanti.

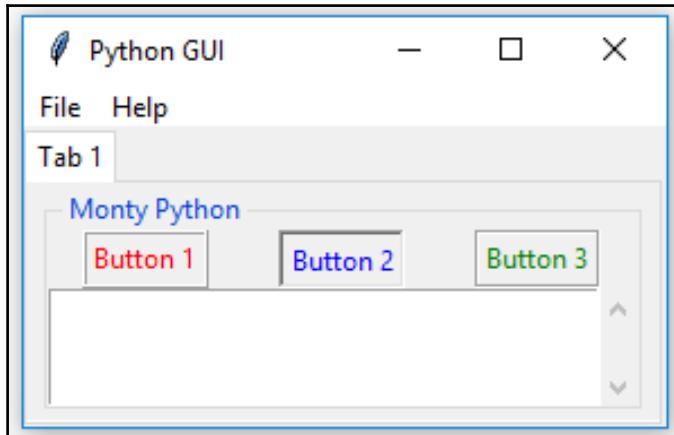
Gli articoli in buttonTypes list sono i nomi delle nostre sottoclassi.



Invochiamo il createButton metodo e quindi chiamare immediatamente il getButtonConfig metodo della classe base e recuperare le proprietà di configurazione utilizzando la notazione a punti.

Quando eseguiamo l'intero codice, otteniamo il seguente Python GUI:

GUI_DesignPattern.py



Possiamo vedere che la nostra fabbrica della GUI Python ha effettivamente creato pulsanti diversi, ognuno con uno stile diverso. Differiscono nel colore del loro testo e nella lorosollievo proprietà.

Come funziona...

In questa ricetta, abbiamo utilizzato il modello di progettazione di fabbrica per creare diversi widget con stili diversi. Possiamo facilmente utilizzare questo modello di progettazione per creare intere GUI.

I modelli di progettazione sono uno strumento molto interessante nella nostra cassetta degli attrezzi di sviluppo software.

Evitare la complessità

In questa ricetta, estenderemo la nostra GUI Python e impareremo come gestire la complessità sempre crescente dei nostri sforzi di sviluppo software.

I nostri colleghi e clienti adorano le GUI che creiamo in Python e chiedono sempre più funzionalità da aggiungere alla nostra GUI.

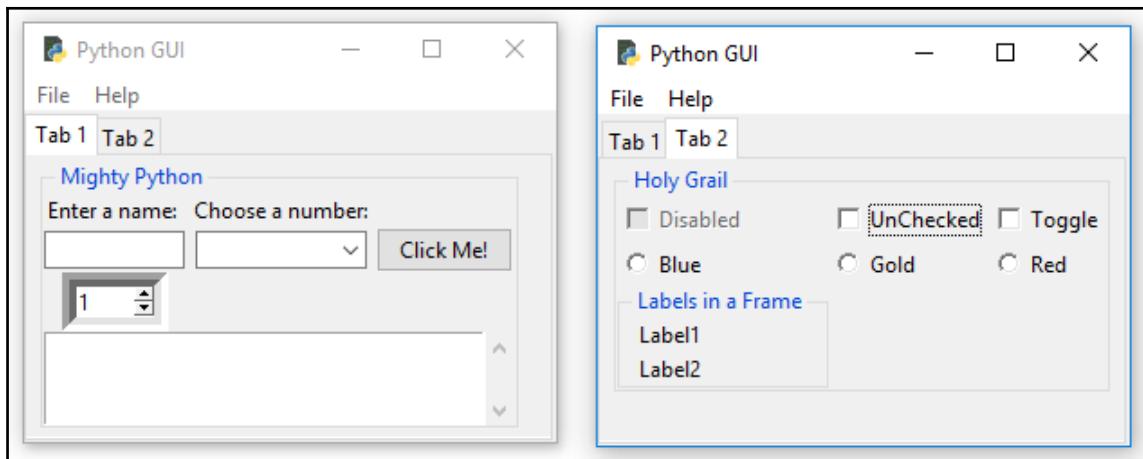
Ciò aumenta la complessità e può facilmente rovinare il nostro bel design originale.

Prepararsi

Creeremo una nuova GUI Python simile a quelle delle ricette precedenti e le aggiungeremo molte funzionalità sotto forma di widget.

Come farlo...

Inizieremo con una GUI Python che ha due schede e che appare come segue. In esecuzione `GUI_Complexity_start.py` risulta quanto segue:



La prima nuova richiesta di funzionalità che riceviamo è quella di aggiungere funzionalità a **Scheda 1**, che cancella il testo scorrevole aggeggio.

Abbastanza facile. Aggiungiamo solo un altro pulsante a**Scheda 1**:

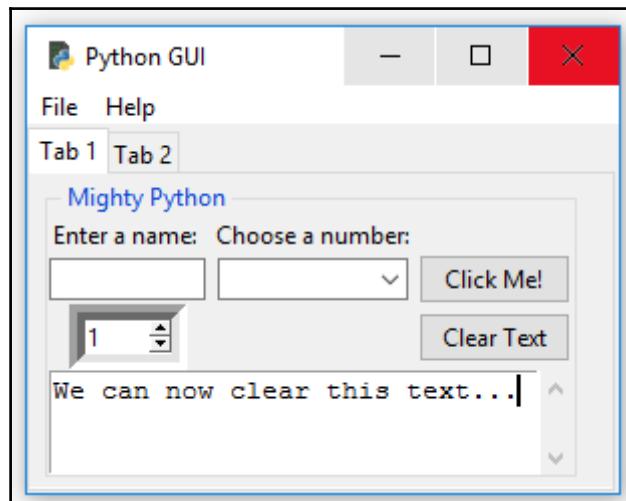
```
# Aggiunta di un altro pulsante
self.action = ttk.Button(self.monty, text="Cancella testo",
command=self.clearScrol)
self.action.grid(colonna=2, riga=2)
```

Dobbiamo anche creare il metodo callback per aggiungere la funzionalità desiderata, che definiamo verso l'alto della nostra classe e al di fuori del metodo che crea i nostri widget:

```
# Pulsante di richiamata
def clickMe(self):
    self.action.configure(text='Hello ' + self.name.get())

# Pulsante callback Cancella testo Clear
def clearScrol(self):
    self.scr.delete('1.0', tk.END)
```

Ora, la nostra GUI ha un nuovo pulsante e, quando lo clicchiamo, cancelliamo il testo del nuovo Testo scorrevole aggiungendo. In esecuzioneGUI_Complexity_start_add_button.py ci dà questo pulsante:



Per aggiungere questa funzionalità, abbiamo dovuto aggiungere codice in due posti nello stesso modulo Python.

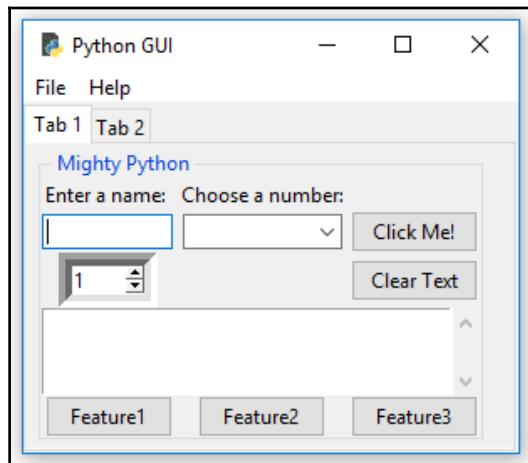
Abbiamo inserito il nuovo pulsante nel creare widget metodo (non mostrato) e quindi abbiamo creato un nuovo metodo di callback, che il nostro nuovo pulsante chiama quando viene cliccato. Abbiamo inserito questo codice appena sotto il callback del nostro primo pulsante.

La nostra prossima richiesta di funzionalità è di aggiungere più funzionalità. La logica di business è incapsulata in un altro modulo Python. Invochiamo questa nuova funzionalità aggiungendo altri tre pulsanti a

Scheda 1. Usiamo un ciclo per fare questo:

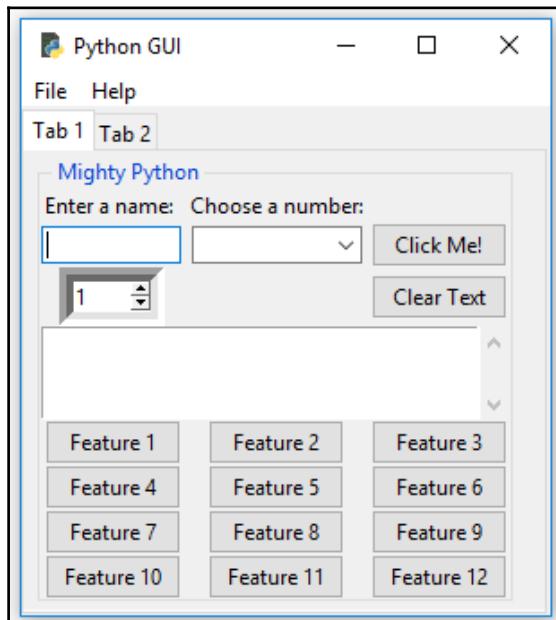
```
# Aggiunta di più pulsanti funzione
per idx nell'intervallo (3):
    b = ttk.Button(self.monty, text="Feature" + str(idx+1))
    b.grid(columna=idx, riga=4)
```

La nostra GUI ora ha questo aspetto, post in esecuzione
GUI_Complexity_start_add_three_more_buttons.py:



Successivamente, i nostri clienti richiedono più funzionalità e utilizziamo lo stesso approccio. La nostra GUI ora ha il seguente aspetto:

GUI_Complexity_start_add_three_more_buttons_add_more.py





Questo non è male. Quando riceviamo nuove richieste di funzionalità per altre 50 nuove funzionalità, iniziamo a chiederci se il nostro approccio sia ancora l'approccio migliore da utilizzare...

Un modo per gestire la complessità sempre crescente che la nostra GUI deve gestire è aggiungere schede. Aggiungendo più schede e inserendo le funzionalità correlate nella propria scheda, otteniamo il controllo della complessità e rendiamo la nostra GUI più intuitiva.

Ecco il codice, `GUI_Complexity_end_tab3.py`, che crea il nostro nuovo **Scheda 3**:

```
# Tab Control 3 -----tab3 = ttk.Frame(tabControl) # Aggiungi
una scheda
tabControl.add(tab3, text='Tab 3') # Rende visibile la tabulazione

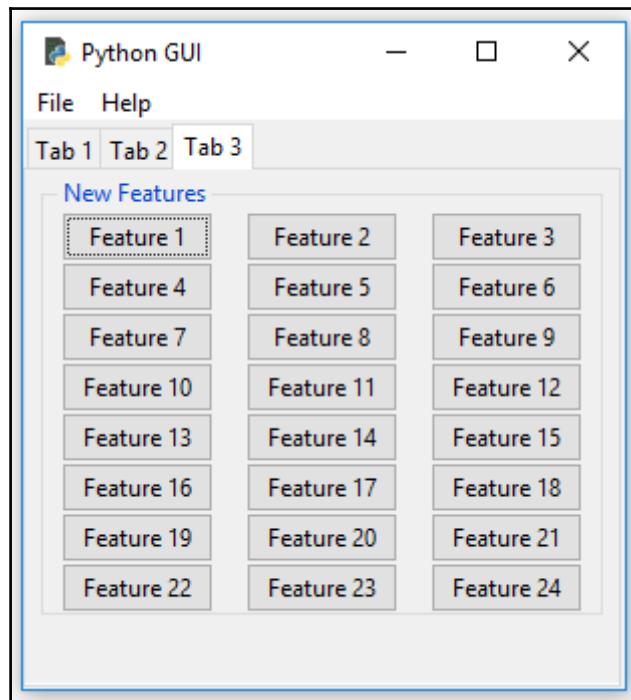
monty3 = ttk.LabelFrame(tab3, text='Nuove funzionalità')
monty3.grid(column=0, row=0, padx=8, pady=4)

# Aggiunta di più pulsanti funzione
riga iniziale = 4
per idx nell'intervallo(24):
    se idx < 2:
        colIdx = idx
        col = colIdx
    altro:
        col += 1
    se non idx % 3:
        riga iniziale += 1
        col = 0

    b = ttk.Button(monty3, text="Feature " + str(idx+1))
    b.grid(column=col, row=riga iniziale)

# Aggiungi dello spazio attorno a ciascuna etichetta
per bambino in monty3.winfo_children():
    child.grid_configure(padx=8)
```

L'esecuzione del codice precedente fornisce la seguente nuova GUI Python:



Come funziona...

In questa ricetta, abbiamo aggiunto diversi nuovi widget alla nostra GUI per aggiungere più funzionalità alla nostra GUI Python. Abbiamo visto come sempre più nuove richieste di funzionalità hanno portato facilmente il nostro bel design della GUI in uno stato in cui è diventato meno chiaro come utilizzare la GUI.

Improvvisamente, i widget hanno conquistato il mondo...



Abbiamo visto come gestire la complessità modularizzando la nostra GUI suddividendo le funzionalità di grandi dimensioni in parti più piccole e disponendole in aree funzionalmente correlate utilizzando le schede.

Sebbene la complessità abbia molti aspetti, la modularizzazione e il refactoring del codice sono in genere un ottimo approccio alla gestione della complessità del codice software.

Progettazione della GUI utilizzando più notebook

In questa ricetta, creeremo la nostra GUI utilizzando più notebook. Sorprendentemente, tkinter non viene spedito immediatamente con questa funzionalità, ma possiamo facilmente progettare un tale widget.

L'utilizzo di più taccuini ridurrà ulteriormente la complessità discussa nella ricetta precedente.

Prepararsi

Creeremo una nuova GUI Python simile a quella della ricetta precedente. Questa volta, tuttavia, progetteremo la nostra GUI con due notebook. Per concentrarci su questa caratteristica, useremo le funzioni invece dei metodi di classe.

Leggere la ricetta precedente sarà una buona introduzione a questa ricetta.

Come farlo...

Per utilizzare più notebook all'interno della stessa GUI, iniziamo creando due frame. Il primo frame conterrà i taccuini e le relative schede, mentre il secondo frame fungerà da area di visualizzazione per i widget che ciascuna scheda è progettata per visualizzare.

Usiamo il gestore del layout della griglia per disporre i due frame, posizionandoli uno sopra l'altro. Quindi, creiamo due taccuini e li sistemiamo all'interno del primo fotogramma:

```
④ # -----
#   Create GUI
#
win = tk.Tk()                      # Create instance
win.title("Python GUI")    # Add title
#
# -----
win_frame_multi_row_tabs = ttk.Frame(win)
win_frame_multi_tabs.grid(column=0, row=0, sticky='N')
display_area = ttk.Labelframe(win, text=' Tab Display Area ')
display_area.grid(column=0, row=1, sticky='WE')
note1 = ttk.Notebook(win_frame_multi_tabs)
note1.grid(column=0, row=0)
note2 = ttk.Notebook(win_frame_multi_tabs)
note2.grid(column=0, row=1)
```

Successivamente, utilizziamo un ciclo per creare cinque schede e aggiungerle a ciascun notebook:

```
# create and add tabs to Notebooks
for tab_no in range(5):
    tab1 = ttk.Frame(note1, width=0, height=0) # Create a tab for notebook 1
    tab2 = ttk.Frame(note2, width=0, height=0) # Create a tab for notebook 2
    note1.add(tab1, text=' Tab {} '.format(tab_no + 1)) # Add tab notebook 1
    note2.add(tab2, text=' Tab {} '.format(tab_no + 1)) # Add tab notebook 2
```

Creiamo una funzione di callback e associamo l'evento click dei due notebook a questa funzione di callback. Ora, quando l'utente fa clic su qualsiasi scheda appartenente ai due notebook, verrà chiamata questa funzione di callback:

```
# bind click-events to Notebooks
note1.bind("<ButtonRelease-1>", notebook_callback)
note2.bind("<ButtonRelease-1>", notebook_callback)
```

Nella funzione di callback, aggiungiamo la logica che decide quali widget vengono visualizzati dopo aver fatto clic su una scheda:

```
def notebook_callback(event):
    clear_display_area()

    current_notebook = str(event.widget)
    tab_no = str(event.widget.index("current")) + 1

    if current_notebook.endswith('notebook'):
        active_notebook = 'Notebook 1'
    elif current_notebook.endswith('notebook2'):
        active_notebook = 'Notebook 2'
    else:
        active_notebook = ''

    if active_notebook is 'Notebook 1':
        if tab_no == '1': display_tab1()
        elif tab_no == '2': display_tab2()
        elif tab_no == '3': display_tab3()
        else: display_button(active_notebook, tab_no)
    else:
        display_button(active_notebook, tab_no)
```

Aggiungiamo una funzione che crea un'area di visualizzazione e un'altra funzione che pulisce l'area:



Nota come la funzione di callback chiama il clear_display_area() funzione.

```
#-----
@def create_display_area():
    # add empty label for spacing
    display_area_label = tk.Label(display_area, text="", height=2)
    display_area_label.grid(column=0, row=0)

#-----
@def clear_display_area():
    # remove previous widget(s) from display_area:
    for widget in display_area.grid_slaves():
        if int(widget.grid_info()["row"]) == 0:
            widget.grid_forget()
```

Il clear_display_area() la funzione conosce sia la riga che la colonna in cui vengono creati i widget delle schede e, trovando la riga zero, possiamo quindi utilizzare grid_forget() per cancellare il display.

Per le schede da 1 a 3 del primo taccuino, creiamo nuove cornici per contenere più widget. Facendo clic su una di queste tre schede si ottiene una GUI molto simile a quella che abbiamo creato nella ricetta precedente.

Queste prime tre schede vengono richiamate nella funzione di callback come display_tab1(), display_tab2(), e display_tab3() quando si fa clic su tali schede.

Ecco il codice che viene eseguito quando si fa clic su **Scheda 3** del primo quaderno:

```
#-----
@def display_tab3():
    monty3 = ttk.LabelFrame(display_area, text=' New Features ')
    monty3.grid(column=0, row=0, padx=8, pady=4)

    # Adding more Feature Buttons
    startRow = 4
    for idx in range(24):
        if idx < 2:
            colIdx = idx
            col = colIdx
        else:
            col += 1
        if not idx % 3:
            startRow += 1
            col = 0

        b = ttk.Button(monty3, text="Feature " + str(idx + 1))
        b.grid(column=col, row=startRow)

    # Add some space around each label
    for child in monty3.winfo_children():
        child.grid_configure(padx=8)
```

Facendo clic su qualsiasi scheda diversa dalle prime tre schede del notebook si richiama la stessa funzione, `display_button()`, che si traduce nella visualizzazione di un pulsante la cui proprietà del testo viene impostata per mostrare il taccuino e il numero di scheda:

```
#-
@def display_button(active_notebook, tab_no):
    btn = ttk.Button(display_area, text=active_notebook + ' - Tab ' + tab_no, \
                     command= lambda: showinfo("Tab Display", "Tab: " + tab_no) )
    btn.grid(column=0, row=0, padx=8, pady=8)
```

Facendo clic su uno di questi pulsanti viene visualizzata una finestra di messaggio.

Alla fine del codice, invochiamo il `display_tab1()` funzione. Al primo avvio della GUI, i widget di questa scheda sono quelli che vengono visualizzati nell'area di visualizzazione:

```
# bind click-events to Notebooks
note1.bind("<ButtonRelease-1>", notebook_callback)
note2.bind("<ButtonRelease-1>", notebook_callback)

create_display_area()

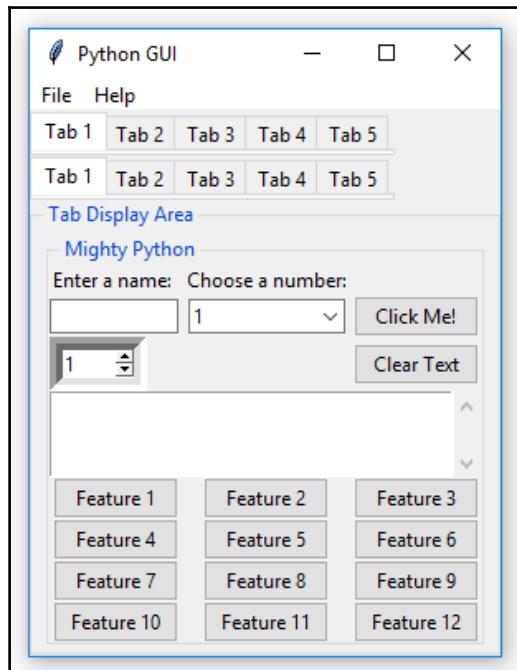
create_menu()

display_tab1()

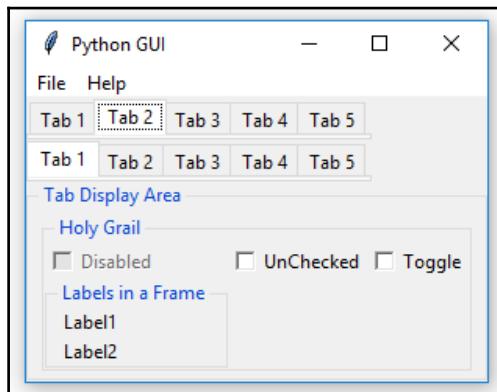
#-----
win.mainloop()
#-----
```

Come funziona...

correndo il `GUI_Complexity_end_tab3_multiple_notebooks.py` codice di questa ricetta crea la seguente GUI:

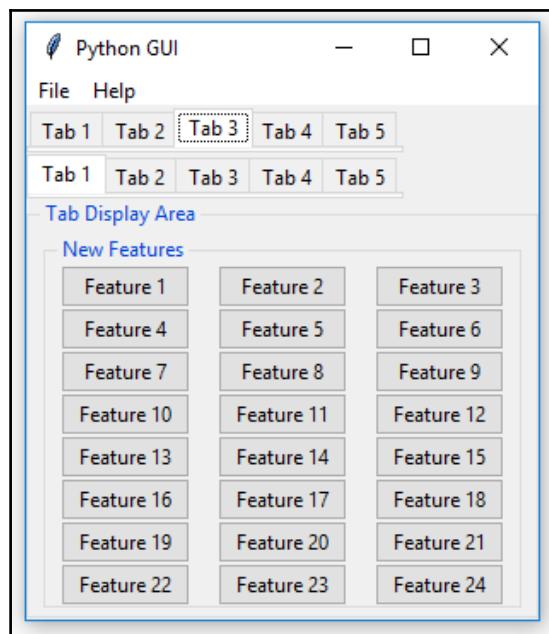


Cliccando su **Scheda 2** del primo taccuino cancella l'area di visualizzazione delle schede e quindi visualizza i widget creati nel display_tab2() funzione:

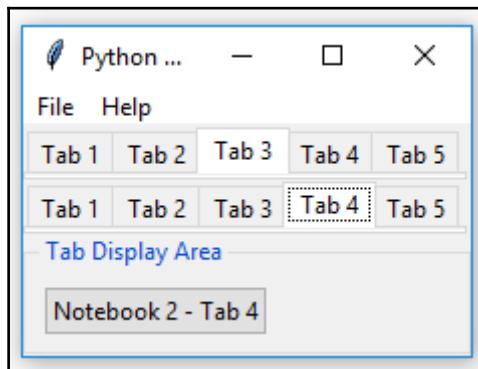


Nota come l'area di visualizzazione della scheda si adatta automaticamente alle dimensioni dei widget che vengono creati.

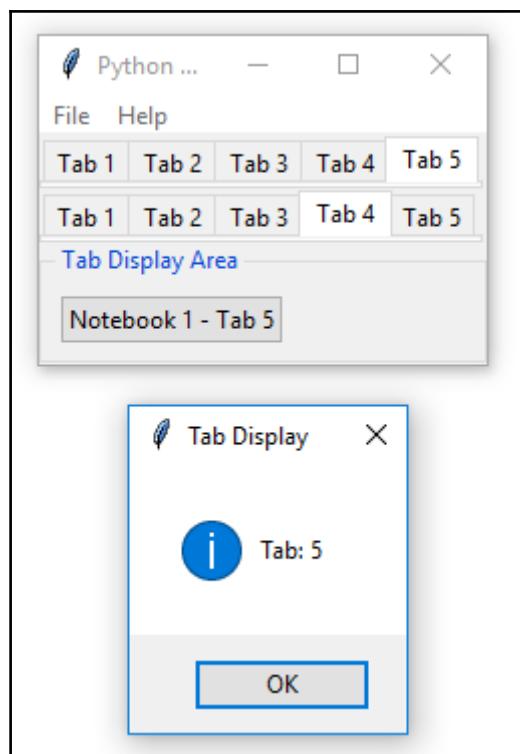
Facendo clic su **Scheda 3** risultati nella seguente visualizzazione della GUI:



Facendo clic su qualsiasi altra scheda nel primo o nel secondo blocco appunti viene visualizzato un pulsante nell'area di visualizzazione della scheda:



Facendo clic su uno di questi pulsanti viene visualizzata una finestra di messaggio:



In questa ricetta, hai imparato come creare più di un notebook all'interno dello stesso design della GUI.

Non c'è limite alla creazione di taccuini. Possiamo creare tutti i taccuini richiesti dal nostro design.



Nella programmazione, in certi momenti, ci imbattiamo in un muro e rimaniamo bloccati.

Continuiamo a sbattere la testa contro questo muro ma non succede nulla. A volte, ci sentiamo come se volessimo arrenderci.

I miracoli accadono...

Continuando a sbattere contro questo muro, a un certo momento, il muro crollerà e la strada sarà aperta.

A quel punto, possiamo incidere positivamente sull'universo del software...

Indice

3

3D

GUI personalizzata, creando con wxPython 339

—

dentro

utilizzato, per il collegamento di moduli 377

sezione principale

utilizzato, per la creazione di codice di autodiagnistica 284, 289

B

bitmap

usato, per creare la GUI 345

C

funzioni di richiamata

scrivere 120, 121, 122

widget di tela

usando 97, 98, 99

grafico

creare 138, 139

creazione, Matplotlib utilizzato 128, 129, 130, 131

etichette, posizionamento 140, 143, 144, 145

legenda, aggiungendo 146, 148

scala, regolazione automatica 152, 153, 154, 155, 156

ridimensionamento 149, 151, 152

pulsante di spunta

creando, con diversi stati iniziali 26

classi

codifica, per migliorare la GUI 114, 115, 119, 120

convenzione di denominazione in codice

usando 388

widget della casella combinata

aggiungendo 24, 25, 26

complessità

evitando 399

Tempo universale coordinato (UTC) 261

Crea, leggi, aggiorna, elimina (CRUD) 203

GUI personalizzata

colori, aggiungendo 351

creando, con wxPython 339

D

demone 169

dati

ottenere, da widget 108, 109

lettura, da siti web con urlopen 197, 201

recupero, dal database MySQL 235, 239

memorizzazione, dal database MySQL 235, 239

eseguire il debug dei livelli di output

configurazione 280, 283

eseguire il debug degli orologi

ambientazione 275

modelli di progettazione

usando 396

widget di dialogo

usato, per copiare file in rete 183, 193

Non ripeterti (DRY) 30, 123

E

Ora legale orientale (EDT) 262

Eclipse PyDev IDE

usato, per scrivere unit test 293, 299

Eclipse, con link di riferimento

al plugin PyDev 279

Widget di ingresso 19, 20

la gestione delle eccezioni

link di riferimento 198

F

cadere

e codice OOP, mescolando 384

Primo in prima uscita (FIFO) 173

infissi

di 294

Riferimenti 294

Telaio 312

G

GLCanvas update

URL, per esempio 337

gestore del layout della griglia

usando 68

Linguaggio della GUI

modificando 252, 255

Widget GUI GUI

allineamento, con frame incorporati all'interno di frame 49,

50, 51, 52, 53, 54

gui2py

URL 345

GUI

codice, test 271, 275

comunicare 327

creazione, più taccuini utilizzati 405

creazione, framework di sviluppo della GUI PyGlet

utilizzato 348

creazione, in OpenGL 335

creazione, unit test utilizzati 289, 293

progettare, in modo agile 267, 271

localizzazione 257

preparazione, per l'internazionalizzazione 263

io

creazione di finestre di messaggio

indipendenti independent 77, 78, 79, 80

Ambienti di sviluppo integrati (IDE) 275

Comunicazione tra processi (IPC) 158, 302

I

Etichetta 50

widget cornice etichetta

etichette, arrangiamento 38, 39, 40, 41

etichette

aggiungendo, al modulo GUI 14

organizzazione, nel widget della cornice dell'etichetta 38, 39, 40, 41

piazzamento, in classifica 140, 143, 144, 145

Last In First Out (LIFO) 173

leggenda

aggiungendo, al grafico 146, 148

ciclo continuo

diversi widget, aggiungendo 34

M

finestra principale principale

icona, modifica 82, 83

Matplotlib

installazione, pip utilizzato con l'estensione whl 131, 132, 133, 134, 136, 137

URL 128, 130

URL, per il download 134

utilizzato, per la creazione di grafici 128, 129, 130, 131

barre dei menu

creare 54, 56, 57, 59, 60, 61, 62

caselle di messaggio

creare 72, 73, 74, 75, 76

potente 50

variabili globali a livello di modulo

usando 110, 112, 113, 114

moduli

code, passaggio 179, 182

più taccuini

uso, per creare la GUI 405

Database MySQL

connessione, configurazione 210, 214

dati, recupero 235, 239

dati, memorizzazione 235, 239

Server MySQL

connessione, da Python 204

installazione, da Python 204

URL, per l'installazione 204

MySQL Workbench

di 239

Riferimenti 241

usando 239, 245

oh

Programmazione orientata agli oggetti

(OOP) about 91

evitando 392

codice, mescolando con fall-down 384

OpenGL

animazione 355
URL 362

P

imbottitura
usato, per aggiungere spazio intorno ai widget 41, 42, 43, 44
Versione Phoenix 303
modulo pip
usato, per installare Matplotlib con estensione whl 131, 132, 133, 134, 136, 137
barra di avanzamento
aggiungendo, alla GUI 94, 95, 96, 97
Framework di sviluppo della GUI PyGlet
utilizzato, per la creazione di GUI 348
Piglet
 URL 350
PyOpenGL 3.0.2
 URL 336
PyOpenGL
 di 333
 importazione 335
Python 3.6
 URL 9
Collegamento di riferimento del
driver del connettore Python 207
Proposta di miglioramento Python (PEP) 373
 URL 9
URL dei pacchetti di estensione
 Python 336
Database della GUI Python
 progettare 222
GUI Python
 pulsanti, creazione 16, 18
 pulsanti, utilizzati per modificare le proprietà del testo 16, 18
 creare 9, 10
 database, progettazione 215
 etichetta, aggiunta 14
 ridimensionare, prevenire 12
Pitone
 Server MySQL, connessione a 204
 Server MySQL, installazione 204
 utilizzato, per controllare diversi framework GUI 324
 usato, per creare tooltip 90, 91, 92, 93, 94

Q

code
passando, tra i moduli 179, 182
usando 173, 179

R

widget del pulsante di opzione
 usando 28, 29, 30, 31
Modalità colore rosso, verde, blu, alfa (RGBA) 356
regressione 275
proprietà di rilievo
 usando 87, 88, 89, 90
componenti GUI riutilizzabili
 creare 122, 123, 124, 125

S

SCHEMI 244
widget di testo scorrevole
 usando 31, 32, 33
codice di autodiagnosi
 creazione, __main__ sezione utilizzata 284, 289
Presentazione
 creazione, tkinter utilizzato 362
codice spaghetti
 evitando 370
controllo della casella di selezione
 usando 83, 84, 85, 86, 87
Comando SQL DELETE
 usando 231, 235
comando SQL INSERT
 usando 222, 225
comando SQL UPDATE
 usando 225, 230
Ora solare (EST) 262
StringVar()
 usando 102, 105, 106, 107

T

widget a schede
 creare 62, 63, 64, 65, 66, 67
tcl
 URL 31
TCP/IP
 usando, per comunicare attraverso le reti 194

Sviluppo guidato dai test (TDD) 293

test

di 271

Codice GUI 271, 275

widget casella di testo text

di 19, 20

disabilitando 21, 22, 23

messaggio a fuoco, impostazione 21, 22, 23

discussioni

creare 159, 163

link di riferimento 158

di partenza 163, 169

fermandosi 169, 172

codice GUI tkinterinter

incorporamento, nella GUI di wxPython 321

tkinter widgetinter

link di riferimento 68

modulo finestra tkinter

titolo, creazione 81

tkinter

usato, per creare slideshow 362

app wxPython, incorporamento 318

suggerimenti

creazione, Python utilizzato 90, 91, 92, 93, 94

tu

test unitari

usato, per creare la GUI 289, 293

scrittura, Eclipse PyDev IDE utilizzato 293, 299

Convenzione di denominazione universale (UNC) 189

urlopen

utilizzato, per la lettura di dati da siti web 197, 201

W

whl estensione

Matplotlib, installazione di pip usata 131, 132, 133, 134, 136, 137

testo del widget

visualizzazione, in diverse lingue 251

widget

GUI, in espansione 44, 46, 47, 49

testo, visualizzazione in diverse lingue 249

vincere 50

wxDesigner

URL 317

app wxPython

incorporamento, nell'app tkinter 318

GUI di wxPython

controlli, aggiungendo 312

GUI, creazione 306

codice GUI tkinter, incorporamento 321

URL 307

libreria wxPython

installazione 303

URL 303

URL, per Phoenix Project 305