Full Circle

LA RIVISTA INDIPENDENTE PER LA COMUNITÀ LINUX UBUNTU

EDIZIONE SPECIALE SERIE PROGRAMMAZIONE

EDIZIONE SPECIALE
SERIE PROGRAMMAZIONE

python™

PROGRAMMARE IN PYTHON
VOLUME 6

Cos'è Full Circle

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

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

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

Full Circle

LA RIVISTA INDIPENDENTE PER LA COMUNITÀ LINUX UBUNTU

Ecco a voi un altro 'Speciale monotematico'

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

Quella che avete davanti è la ristampa della serie '**Programmare in Python' parti 32-38,** pubblicata nei numeri 60-67.

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

Buon divertimento!

Come contattarci

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

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

IRC:#fullcirclemagazine su chat.freenode.net

Gruppo editoriale

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

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

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

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

Iniziare Python - Parte 32

Scritto da Greg D. Walters

Devo dire che amo il mio tablet Android. Sebbene lo usi tutti i giorni, non è ancora un rimpiazzo per il mio computer. E devo anche ammettere, molto dell'uso che ne faccio è pressapoco lo stesso di chiunque altro: navigare sul web, ascoltare musica, guardare video, giocare e cosi via. Provo solo a giustificarlo avendo alcuni applicazioni che hanno a che fare con la lista della spesa e delle cose da fare, per trovare distributori di benzina economici, cose divertenti per il nipote etc. Fin ora è veramente un giocattolo per me. Perché usare un simpatico schermo sensibile al tocco per la lista della spesa? Ammettiamolo... è per gli sguardi freddi di invidia che le persone mi fanno quando mi vedono spingere il carrello lungo il corridoio toccando il mio tablet per smarcare gli elementi dalla lista. Ahh.. il fattore geek comanda! Naturalmente, posso usare il retro di una vecchia busta per tenere la mia lista. Ma questo non sarebbe altrettanto figo e geek, ora, vero?

Come per il 99% dei mariti geek nel mondo, sono sposato con una donna non geek. Una donna amorevole e meravigliosa, per essere sicuro, ma non una geek e quando comincio a sbavare per l'ultimo aggeggio, sospira e dice qualcosa tipo" Bene, se veramente pensi che abbiamo bisogno di questo ...". Poi mi da lo stesso sguardo che le do io quando sta amorevolmente accarezzando il cinquantesimo paio di scarpe al negozio. In tutta onestà, non è stato difficile portare il primo tablet in casa nostra. Lo portai per mia moglie quando stava facendo la chemioterapia. Per un po' ha cercato di usare un portatile, ma il calore e il peso sul suo grembo era troppo dopo un po'. Gli e-book sul computer portatile per lei non erano una opzione, così quando tentava di leggere doveva destreggiarsi tra il libro, il portatile e il lettore mp3. Tutto questo mentre era legata ad un poltrona con i tubi infilati nel braccio che la stavano riempendo di sostanze chimiche. Quando le portai il tablet, fu la soluzione ideale. Poteva leggere un e-book, ascoltare musica, guardare un programma televisivo, navigare sul web, controllare la sua posta elettronica, aggiornare il suo blog sul cancro, seguire i suoi amici su facebook e giocare, il tutto su un dispositivo leggero e favoloso. Se si stancava, le bastava farlo scivolare di lato tra lei e il lettino reclinabile (o il letto quando era a casa cercando di recuperare le forze). MOLTO meglio di un portatile ingombrante, un libro, un lettore mp3, un telecomando e altro ancora.

Mentre si riempiva di sostanze chimiche nocive, ho voluto impadronirmi di una sedia e di un tavolo nell'angolo della stanza del trattamento, vicino ad una presa di corrente e provare a lavorare con mio portatile vecchio di sei anni. Tra i progetti, volevo fare qualche ricerca sulla programmazione Android. Avevo scoperto che la maggior parte della programmazione è fatta in Java. Mi ero quasi rassegnato a imparare nuovamente Java quando mi imbattei in un paio di strumenti che permettono la programmazione Python per il sistema operativo Android. Uno di questi strumenti è chiamato "SL4A", che sta per Strato di Scripting per Android. Su questo concentreremo i prossimi due articoli. In questo articolo ci focalizzeremo sul configurare SL4A su Android.

Potreste chiedere perché mai vorrei parlare di programmazione Android su una rivista progettata per Linux. Bene, la ragione semplice è che il cuore di Android è Linux. Tutto ciò che è Android si trova sopra a Linux!

Molte pagine web mostrano come caricare SL4A nell'emulatore Android per Desktop. Vedremo come farlo un'altra volta, per ora ci occuperemo del dispositivo Android stesso. Per installare SL4A sul proprio dispositivo Android, andate presso http://code.google.com/p/androidscripting/; troverete il file di installazione per SL4A. Non siate assolutamente

confusi. C'è un codice a barre quadrato ad alta densità che dovete premere per scaricare l'APK. Assicuratevi di avere l'opzione " origini sconosciute" attivata nelle Impostazioni della Applicazioni. Si tratta di uno scarico veloce. Uno volta che lo avete scaricato e installato, andate avanti, trovate l'icona e premetela. Quello che vedrete è una schermata nera piuttosto deludente che dice "Scripts...No matches found". Questo va bene. Premete il pulsante menu e selezionate Vista. Vedrete un menù. Selezione Interpreti. Quindi selezionate di nuovo menù e poi Add (Aggiungi). Dal menù successivo selezionate Python2.6.2. Vi dovrebbe chiedere di avviare una sessione del browser per scaricare Python per Android. Una volta installato, selezionate Open (Apri). Otterrete una schermata di menù con le opzioni Install (Installare), Import Modules (Importare Moduli), Browse Modules (sfoglia moduli) e Uninstall Modules (Disinstalla Moduli). Ora python scaricherà e installerà altri moduli aggiuntivi. In aggiunta otterrete alcuni esempi di script. Infine premete il pulsante indietro e vederete Python 2.6.2 installato negli interpreti dello schermo. Toccate ancora sul pulsante back (indietro) e vedrete una lista di alcuni script di esempio python.

> **vedrete una schermata nera piuttosto deludente [...] Questo va bene.**

Questo è tutto quello che faremo questa volta. Ciò che voglio è stimolare il vostro appetito. Esplorate Python per Android. Potreste anche voler visitare http://developer.android.com/sdk/index.html per ottenere l'SDK di Android (il Kit di Sviluppo Software) per il vostro desktop. Include un emulatore Android in modo da poterci giocare a lungo. Configurare l'SDK su Linux è davvero abbastanza facile, quindi non dovreste avere molti problemi.

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

Come includere Accenti dalla tastiera

di Barry Smith

Se il vostro sistema Linux è in francese, tedesco o spagnolo e quindi richiede accenti, o occasionalmente avete bisogno di accenti che non appaiono nella parole inglese, molti utenti non sanno che c'è un modo molto semplice per farlo dalla tastiera. Le seguenti combinazioni si applicano solo alla tastiera UK.

Accento acuto
Premi AltGr +; (punto e virgola) alza la mano e quindi premi la vocale desiderata é

Accento circonflesso
Premi AltGr +' (apostrofo) alza la mano e quindi premi la vocale desiderata î

Accento grave
Premi AltGr +# (cancelletto) alza la mano e quindi premi la vocale desiderata è

Dieresi
Premi Alt Gr + [Alza la mano quindi premi u ü

ñ - Premi Alt Gr +] Alza la mano quindi premi n ñ

oe- Premi Shift + Alt Gr Alza la mano quindi premi o quindi premi e oe Il carattere oe non apparirà fino a che e è premuta.

Per ottenere ¿ e ¡ (punti esclamativi invertiti) che uso sempre in spagnolo prima delle domande e delle esclamazioni, premi ?AltGr + Maiusc,tenendo entrambi i tasti premuti, premi quindi _ (sottolineato) per ¿ o premi ! (punto esclamativo) per ¡.

Se vuoi qualcuna di queste in maiuscolo, premi Maiusc prima di digitare la lettera.

Iniziare Python - Parte 33

Questa volta configureremo l'SDK di Android sul vostro desktop Linux. Creeremo inoltre un dispositivo virtuale Android, installeremo SL4A e python su questo e faremo un piccolo test.

Per favore siate consapevoli, questo non è qualcosa che voi dovreste fare per macchine che hanno meno di 1 GB di ram. L'emulatore si mangia una grande quantità di memoria. Ho provato su un portatile su cui stava girando Ubuntu con 512 MB di ram. FUNZIONERA' ma è VERAMENTE lento.

Qui c'è una lista di quello che faremo. Andremo passo passo in un minuto.

- Installare il JDK6 di Java.
- Installare il pacchetto per iniziare con Android SDK.
- Creare e installare AVD.
- Testare AVD, installare SL4A e Python

In realtà, dovremmo anche installare Eclipse e il plugin Android ADT per Eclipse, ma, siccome non vogliamo avere a che fare con Eclipse in questo insieme di articoli, possiamo bypassare questo. Se volete includere questi passi, date una occhiata a http://developer.android.com/sdk/installing.html per vedere tutti i passi nell'ordine suggerito. Partiamo.

PASSO 1 – Java JDK 6

Da tutto ciò che ho letto e provato, deve essere veramente il rilascio SUN. Open JDK si suppone che non funzioni. Potete trovare informazioni a proposito di ciò nel web, ma qui ci sono i passi che ho seguito. In un terminale, digitate i seguenti...

```
sudo add-apt-repository ppa:ferramroberto/java

sudo apt-get update

sudo apt-get install sun-java6-jdk
```

Una volta che è stato fatto tutto, dovrete modificare il tuo file bashrc per assegnare la variabile "JAVA_HOME" in modo che tutto funzioni correttamente. Ho usato gedit e alla fine del file ho aggiunto la seguente linea

```
export JAVA_HOME="/usr/lib/jvm/java-6-sun1.6.0.06"
```

Salvate il file e procedete al passo 2.

PASSO 2 – ANDROID SDK Pacchetto Iniziale

Ora inizia il divertimento vero e proprio. Dovete andare su developer.android.com/sdk/index.html. Qui è dove si trova l'SDK. Scaricate l'ultima versione per Linux che, al momento in cui sto scrivendo, è androidsdk_r18-linux.tgz. Usando il Gestore degli Archivi, decomprimetelo in un posto conveniente. Io l'ho messo nella mia directory home. Tutto viene eseguito direttamente da questa cartella, così voi non avrete bisogno di installare nient'altro. Così il percorso per me è /home/greg/android-sdk-linux. Navigate in questa cartella, quindi dentro alla cartella tools. Qui troverete un file chiamato "android". Questo è quello che esegue il vero SDK. Ho creato un lanciatore sulla mia scrivania, per renderlo facile da raggiungere.

Ora la parte noiosa. Eseguite il file android e Android SDK Manager partirà. Verificherà e aggiornerà le piattaforme che sono disponibili. Vi avverto ora che questo processo richiederà un certo tempo, così non preoccupatevi se non avete un sacco di tempo per gestirlo. Per brevità vi consigli di iniziare con una sola piattaforma. Una buona per iniziare è Android 2.1, perché, per la maggior parte, se si sviluppa per una una vecchia piattaforma, non ci dovrebbero esser problemi su una piattaforma più nuova. Avete anche bisogno di impostare opportunamente gli Strumenti. Semplicemente selezionate la casella accanto a questi due elementi, quindi fare clic sul pulsante di installazione. Una volta che avete ottenuto la piattaforma di vostra scelta e lo strumento configurato, sarete quasi pronti per creare la vostra prima macchina virtuale.

PASSO 3 – Creare e configurare il vostro primo AVD

Tornando a Android SDK manager, selezionate strumenti dal menu principale, quindi selezionate Managar AVDs. Questo aprirà una nuova finestra. Dal momento che questa è la prima volta, non ci sarà alcun dispositivo virtuale configurato.

Cliccate sul pulsante Nuovo. Questo apre un'altra finestra dove definiamo le proprietà del dispositivo virtuale Android. Qui ci sono i passi per configurare un semplice dispositivo virtuale Android.

• Assegnate il nome del dispositivo. Questo è importante se avete più di un dispositivo.
• Impostate il livello della piattaforma di destinazione
• Impostate le dimensioni della SD (vedere in seguito).
• Impostate la risoluzione dello schermo
• Create il dispositivo

Quindi nella casella di testo Nome, digita "Test1". Nella combobox dell'obiettivo selezionate Android 2.1 – Livello API 7. Nella casella di testo per "SD card! Inserite 512 e assicuratevi che la lista discesa mostri "MiB". Sotto "Skin", impostare la risoluzione a 800x600. (Potete giocare a vostro piacimento con gli altri formati). Finalmente, premete il pulsante "Creare AVD". Presto verrà visualizzato un messaggio dicendo che la AVD è stata creata.

Passo 4 – Testare l'AVD e installare SL4A e Python

Ora, finalmente, possiamo avere un po' di divertimento. Evidenziate l'AVD che avete appena creato e cliccate sul pulsante Partenza, nella finestra di dialogo che spunta fuori. Ora, dovete aspettare un paio di minuti perché il dispositivo virtuale sia creato in memoria e che la piattaforma Android sia caricata e avviata (parleremo di come accelerare questo processo nelle successive esecuzioni).

Una volta che l'AVD si avvia e avete lo schermo "home" davanti a voi, installerete SL4A. Usando il browser o la finestra di ricerca web di Google, sulla schermata iniziale cercate "SL4A". Vai alla pagina dei download, alla fine trovate la pagina per i download su: http://code.google.com/p/androidscripting/downloads/list.

Scorrete la pagina fino a quando non arrivate al collegamento sl4a_r5. Aprite il collegamento e toccate il collegamento del "sl4a_r5.apk". Notate che ho detto "toccate" invece di "cliccate". Cominciate a pensare con il dito per toccare lo schermo, piuttosto che a fare click con il mouse. Renderà più facile la vostra transizione nella programmazione. Vedrete iniziare lo scaricamento. Potrebbe essere necessario tirare giù la barra di notifica per arrivare al file scaricato. Toccate su questo e quindi toccate il pulsante dell'installazione.

Una volta che il file è stato scaricato, vi sarà presentata l'opzione di aprire l'applicazione scaricata o toccare "Fatto" per uscire dal programma di installazione. Qui dovremo toccare "Apri".

Ora SL4A partirà. Probabilmente vedrete una finestra di dialogo che vi chiede se consentite di utilizzare il monitoraggio. O accettate o rifiutate, questo spetta a voi. Prima di andare oltre dovreste conoscere alcune scorciatoie da tastiera che vi aiuteranno negli spostamenti. Dal momento che non avete un reale dispositivo Android, pulsanti come Indietro, Home e Menu non sono disponibili. Avrete bisogno di loro per navigare. Ecco alcuni dei collegamenti importanti.

Indietro-Escape
Home-Home
Menu-F2

Ora dovrete scaricare e installare python in SL4A. Per fare questo, prima toccate menu (premete F2). Selezionate "View" (Visualizza) dal Menu. Ora selezionate "Interpreters"(Interpreti). Sembra che non sia successo niente ma toccate Menu ancora (F2), poi selezionate "Add" (Aggiungere) dal menu a comparsa. Ora scorrete verso il basso e selezionate "Python 2.6.2". Questo scaricherà il pacchetto di base per Python per Android. Installate il pacchetto e poi apritelo. Vi saranno presentate quattro opzioni. Install (Installa), Import Modules (Importa Moduli), Browse Modules (Sfoglia Moduli), Uninstall Module (Disinstalla Moduli). Toccate su Installa. Questo farà partire, scaricare e installare tutti i pezzi della ultima versione di Python per Android. Ciò potrà richiedere qualche minuto.

Una volta che tutto ciò è stato completato, toccate Back (tasto Esc) fino ad arrivare allo schermo degli Interpreti SL4A. Ora tutto è stato caricato per noi per giocare con Python con Android. Toccate Python 2.6.2, e

sarete nella shell Python "standard". Questa è come la vostra shell standard del vostro pc fisso. Digitate le seguenti tre linee una alla volta nella shell. Assicuratevi di aspettare per il prompt ">>>" ogni volta.

```
import android

droid = android.Android()

droid.makeToast("Hello from Python on Android")
```

Dopo aver digitato l'ultima linea e premuto Invio, vedrete un casella con i bordi arrotondati al centro della shell che dice "Hello from Python on Android". Questo è quello che fa il comando "droid.makeToast".

Avete scritto il vostro primo script Python per Android. Pulito, huh?

Ora create un collegamento sulla schermata principale di Android. Toccate la chiave Home (pulsante Home). Se voi scegliete la piattaforma 2.1, dovreste vedere una barra a scorrimento sulla destra dello schermo. Se si sceglie una altra piattaforma, potrebbe essere un quadrato o un rettangolo costituito da piccoli quadrati. Ad ogni modo ottenete la schermata delle Apps (Applicazioni). Toccate questo e trovate l'icona SL4A. Ora eseguite un "tocco lungo", che vi creerà un collegamento sulla schermata iniziale. Spostate il collegamento dove preferite.

Successivamente, creeremo il nostro primo script salvato. Tornate indietro a SL4A. Dovrebbe esservi presentato con gli script di esempio che arrivano con Python per Android. Toccate il pulsante menu e selezionate "Add" (Aggiungi). Selezionate "Python 2.6.2" dalla lista. Verrà presentato l'editor dello script. Nella parte superiore c'è la casella per il nome del file con l'estensione ".py" già compilata. Qui di seguito c'è la finestra dell'editor che ha già inserito per noi le prime due righe del programma. (Ho incluso sotto in corsivo in modo da poter controllare. Abbiamo anche usato queste due linee nel nostro primo esempio).

```
import android

droid = android.Android()
```

Adesso inseriamo le seguenti due righe nello script Python.

```
uname =
droid.dialogGetInput("What's
your name?")

droid.makeToast("Hello %s from
Python on Android") %
uname.result
```

La prima nuova linea creerà una finestra di dialogo (droid.dialogGetInput()) che chiede il nome dell'utente. La risposta è restituita al nostro programma in uname.result. Abbiamo già usato la funzione droid.makeToast().

Nominate il file andtest1.py e quindi toccate Done (Fatto) e toccate "Save & Run". Se tutto ha funzionato, si dovrebbe vedere una finestra di dialogo che chiede il vostro nome. Dopo che lo avete inserito, dovreste vedere un avviso sul fondo dello schermo che dice "Hello, Tuo Nome from Python on Android".

Questo è tutto per questa volta. Per ora, c'è una TONNELLATA di documentazione su SL4A disponibile gratuitamente sul web. Potete giocare un po' da soli fino alla prossima volta. Vi suggerisco di iniziare andando su http://code.google.com/p/androidscripting/wiki/Tutorials.

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

Iniziare Python - Parte 34

Questa volta, finiremo di usare SL4A. Faremo un programma più grande e poi lo spediremo alla macchina virtuale tramite ADB.

Diamoci da fare con il nostro primo codice. In questo, proveremo alcuni dei "widgets" che sono disponibili a noi quando usiamo SL4A. Cominciate dal vostro desktop usando il vostro editor preferito.

Inserite il codice mostrato in alto a destra e salvatelo (ma non provate a eseguirlo) come "atest.py".

```
import android

droid = android.Android()
uname = droid.dialogGetInput("Hello","What's your name?")
droid.makeToast("Hello %s from python on Android!" % uname.result)
```

```
droid.dialogCreateAlert(uname.result,"Would you like to play a game?")
droid.dialogSetPositiveButtonText('Yes')
droid.dialogSetNegativeButtonText('No')
droid.dialogShow()
while True: #wait for events for up to 10 seconds...
      response = droid.eventWait(10000).result
      if response == None:
      break
      if response["name"] == "dialog":
      break
droid.dialogDismiss()
```

La prima linea importa la libreria android. Creiamo una istanza di questa nella seconda linea. La linea 3 crea e mostra una finestra di dialogo con il titolo "Hello", il prompt di "quale è il tuo nome?", una casella di testo per l'utente per inserire il suo nome e due pulsanti "OK" e "Cancel". Una volta che l'utente preme "OK", la risposta è ritornata nella variabile uname. L'ultima linea (fino ad ora) quindi dice "Hello {username} from python on Android!". Questo non è nuovo, abbiamo fatto questo prima. Ora aggiungeremo più codice (sopra).

Salvate il vostro codice come atest1.py. Lo invieremo alla nostra macchina virtuale dopo che avremo discusso cosa fa.

Date una occhiata alle prime quattro linee che abbiamo appena inserito. Abbiamo creato un dialogo di tipo alert chiedendo "Would you like to play a game?" (ndt "Vorresti giocare una partita?"). Nel caso di un dialogo di tipo alert, non c'è nessuna casella per inserire del testo. Le due linee successive dicono di creare due pulsanti, uno con il testo "Yes", che è un pulsante "positivo" e uno con il testo "No" un pulsante "negativo". I pulsanti positivi e negativi si riferiscono alla risposta ritornata – o "positiva" o "negativa". La linea successiva quindi mostra il dialogo. Le successive sette linee aspettano per una risposta dell'utente.

Creiamo un semplice ciclo (while True:) che aspetta per una risposta fino a 10 secondi usando la chiamata droid.eventWait(value). La risposta (sia "positiva" che "negativa") sarà ritornata nella – avete indovinato - variabile di risposta. Se risposta ha il nome del "dialogo", allora usciremo dal ciclo e ritorneremo la risposta. Se non è successo niente prima che il timeout avvenga, semplicemente usciremo dal ciclo. La informazione affettivamente restituita nella variabile di risposta è qualcosa tipo questo (assumendo che il pulsante "positivo" o "Yes" sia stato premuto) ...

```
{u'data': {u'which':
u'positive'}, u'name':
```

```
u'dialog', u'time':
1339021661398000.0}
```

Potete vedere che il valore è passato nel dizionario dati, la chiave del dialogo è nel dizionario "name", e c'è un valore 'time' che non ci interessa qui.

Infine dobbiamo chiudere la finestra di dialogo.

Prima di inviare il nostro codice alla macchina virtuale, dobbiamo far partire la macchina virtuale. Avviate il vostro emulatore Android. Una volta che si è avviato, notate che la barra del titolo ha quattro cifre all'inizio del titolo. Questa è la porta che la macchina sta ascoltando. Nel mio caso (e probabilmente anche nel vostro) è 5554.

Ora spingiamolo nella nostra macchina virtuale. Aprite una finestra di terminale e muovetevi nella cartella in cui avete salvato il codice. Assumendo che abbiate configurato il percorso per includere l'SDK, digitate

```
adb devices
```

Questo chiede a adb di mostrare tutti i dispositivi che sono connessi. Questo può includere non solo l'emulatore, ma anche smartphone, tablet o altri dispositivi Android. Dovreste vedere qualcosa come questo ...

```
List of devices attached
emulator5554      device
```

Ora siamo sicuri che il nostro dispositivo è attaccato, vogliamo spingere la applicazione nel dispositivo. La sintassi è ...

```
adb push source_filename
destination_path_and_filename
```

Così, nel mio caso, sarà ...

```
adb push atest1.py
/sdcard/sl4a/scripts/atest1.py
```

Se tutto funziona correttamente, otterrete un messaggio abbastanza deludente simile a questo ...

```
11 KB/s (570 bytes in 0.046s)
```

Ora sull'emulatore Android, parte SL4A. Dovreste vedere tutti gli script python e tra questi dovreste vedere atest1.py. Toccate (cliccate) su atest1.py, e vedrete un dialogo popup con sei icone. Da sinistra a destra essi sono "Run in a dialog window", "Run outside of a window", "Edit", "Save", "Delete" e "Open in un editor esterno". In questo momento toccate (cliccate) sulla icona più a sinistra "Run in a dialog window" in modo da poter vedere cosa accade.

Vedrete il primo dialogo che chiede il vostro nome. Inserite qualcosa nella casella e toccate (cliccate) il pulsante "OK". Quindi vedrete il dialogo di alert. Toccate (cliccate) su uno dei pulsanti per chiudere la finestra di dialogo. Non stiamo ancora guardando le risposte così non importa quale scegliete. Ora aggiungeremo dell'altro codice (in altro a destra).

Sono sicuro che si può capire che questo pezzo di codice semplicemente controlla la risposta, e, se è "None" a causa di un timeout, semplicemente stampiamo "Timed out". E, se in realtà è qualcosa che vogliamo, allora assegniamo i dati alla variabile rdialog. Ora aggiungiamo il prossimo pezzetto di codice (sotto) ...

```
if response==None:
    print "Timed out."
else:
    rdialog=response["data"]
```

Questa parte del codice guarderà i dati passati indietro dall'evento pulsante-premuto. Controlliamo se la risposta ha un una chiave "which" e, in questo caso, si tratta di una legittima pressione di pulsante per noi. Controlliamo quindi se il risultato è una risposta "positiva" (pulsante 'Ok'). Se è così creeremo una altra finestra di avviso, ma questa volta si aggiungerà un elenco di elementi da cui l'utente potrà scegliere. In questo caso, offriamo all'utente di selezionare da un elenco comprendente Checkers,Chess, Hangman, and

```
    if rdialog.has_key("which"):
    result=rdialog["which"]
    if result=="positive":
        droid.dialogCreateAlert("Play a Game","Select a game to play")
        droid.dialogSetItems(['Checkers','Chess','Hangman','Thermal
Nuclear War']) # 0,1,2,3
        droid.dialogShow()
        resp = droid.dialogGetResponse()
```

Thermal Nuclear War, e assegniamo il valore da 0 a 3 per ogni elemento. (Questo inizio vi sembra familiare? Sì è da un film). Mostriamo quindi la finestra di dialogo e aspettiamo per la risposta. La parte della risposta cui siamo interessati è in forma di un dizionario. Supponendo che l'utente abbia toccato (cliccato) su Chess (ndt Scacchi), la risposta risultante risulta come questa ...

```
Result(id=12,
result={u'item':1},
error=None)
```

In questo caso quello che ci interessa davvero è la porzione del risultato dei dati restituiti. La selezione è #1 ed è tenuta nella chiave 'item'.

```
if resp.result.has_key("item"):
    sel = resp.result['item']
    if sel == 0:
        droid.makeToast("Enjoy your checkers game")
    elif sel == 1:
        droid.makeToast("I like Chess")
    elif sel == 2:
        droid.makeToast("Want to 'hang around' for a while?")
    else:
        droid.makeToast("The only way to win is not to play...")
```

Ecco la parte successiva del codice (in alto a destra) ...

Qui controlliamo se la risposta ha la chiave "item", e, se è così, assegniamola alla variabile "sel". Ora usiamo un ciclo se/altrimenti se /altrimenti per controllare i valori e gestire qualsiasi cosa sia stato selezionato. Usiamo la funzione droid.makeToast per mostrare la risposta. Naturalmente, potete aggiungere il vostro codice qui. Ora per terminare il codice (in basso a destra)

Come potete vedere, semplicemente rispondiamo agli altri tipi di pulsanti premuti qui.

Salvate, spingete e eseguite il programma.

Come potete vedere, SL4A vi da la capacità di fare applicazione "Rese con

```
        elif result=="negative":
            droid.makeToast("Sorry.  See you later.")
    elif rdialog.has_key("canceled"):
        print "Sorry you can't make up your mind."
    else:
        print "unknown response=",response
print "Done"
```

interfaccia grafica", ma non applicazioni completamente grafiche. Questo comunque non dovrebbe trattenervi dall'andare avanti e iniziare a scrivere i vostri programmi per Android. Non aspettatevi di metterli su nel "market". La maggior parte delle persone vogliono applicazioni di tipo completamente grafico. Guarderemo a queste la prossima volta. Per ulteriori informazioni su SL4A, semplicemente fate una ricerca sul web e troverete molti tutorial e maggiori informazioni.

Comunque, potete spingere direttamente sul vostro smartphone o tablet nello stesso modo. Come al solito il codice è stato messo su pastebin a http://pastebin.com/REkFYcSU

Arrivederci alla prossima.

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

Iniziare Python - Parte 35

Scritto da Greg D. Walters

Questa volta, stiamo per prendere un piccola deviazione dalla nostra esplorazione della programmazione Android e guardiamo ad un nuovo framework per programmare GUI chiamato Kivy. Dovrete andare avanti a http://kivy.org e scaricare e installare il pacchetto prima di fare troppo per l'installazione di questo mese. Le istruzioni per l'installazione su Ubuntu possono essere trovate all'indirizzo http://kivy.org/docs/installation/installation-ubuntu.html.

Prima di tutto, Kivy è una libreria open source che fa uso di display multitouch. Se questo non è abbastanza "cool" è anche multipiattaforma, ciò significa che funzionerà su Linux, Windows, Mac OSX, IOS e Android. Ora potete capire perché stiamo parlando di questo. Ma ricordate, per la maggior parte, qualsiasi cosa voi codifichiate usando Kivy, può funzionare su qualsiasi di queste piattaforme senza ricodificare.

Prima di andare troppo avanti, fatemi dire un paio di cose. Kivy è VERAMENTE potente, vi fornisce un nuovo insieme di strumenti per fare la vostra programmazione GUI. Dopo che tutto questo è stato detto, Kivy è anche abbastanza complicato da gestire. Siete limitati ai widget che vi vengono forniti. In aggiunta non c'è alcun strumento GUI per progettare GUI con Kivy, così dovete fare un GRANDE lavoro di pre-pianificazione prima che proviate a fare qualcosa di complicato. Inoltre ricordate che Kivy è continuamente sotto sviluppo, così le cose possono cambiare rapidamente. Finora non ho trovato nessuno del mio codice di prova che è stato rotto da una nuova versione di Kivy, ma c'è sempre una possibilità.

Piuttosto che saltare dentro e creare un nuovo codice questo mese, guarderemo alcuni esempi che sono forniti con Kivy e, il prossimo mese, scriveremo qualcosa di nostro.

Una volta che avete decompresso Kivy nella sua cartella, usate un terminale e andate in quella cartella. La mia è in /home/greg/Kivy-1.3.0. Ora andate nella cartella degli esempi, e quindi nella cartella dei widget. Andiamo a vedere l'esempio accordion_1.py.

È veramente semplice, ma mostra un widget veramente pulito. Sotto c'è il codice.

Come potete vedere, le prime tre righe sono istruzioni di importazione. Qualsiasi widget voi usiate devono essere importati, e dovete sempre importare App da kivy.app.

Le otto righe successive sono la classe principale dell'applicazione. La classe è definita, quindi viene creata una routine chiamata build. Avrete sempre una routine 'build' da qualche parte nei vostri programmi Kivy. Poi assegniamo un oggetto radice dal widget Accordion. Successivamente creiamo cinque Accordionitems e assegniamo il loro titolo. Aggiungiamo poi dieci etichette con il testo "very big content" (ndt "contenuti veramente grandi"). Aggiungiamo quindi, ciascuna etichetta al widget

```
from kivy.uix.accordion import Accordion, AccordionItem
from kivy.uix.label import Label
from kivy.app import App

class AccordionApp(App):
     def build(self):
          root = Accordion()
          for x in xrange(5):
               item = AccordionItem(title='Title %d' % x)
               item.add_widget(Label(text='Very big content\n' * 10))
               root.add_widget(item)
          return root

if __name__ == '__main__':
     AccordionApp().run()
```

radice e infine ritorniamo l'oggetto radice. Questo in sintesi mostra l'oggetto radice nella finestra che Kivy ha creato per noi. Finalmente abbiamo l'istruzione "if __name__" e quindi eseguiamo l'applicazione.

Andiamo avanti e eseguiamo per vedere che cosa fa.

Vedrete in un attimo o due, una finestra che si apre con cinque barre verticali all'interno. Facendo clic su una di esse, fa si che si apra rivelando le dieci etichette. Naturalmente ciascuna barra ha lo stesso testo nelle dieci etichette, ma voi potete capire come aggiustarlo.

Il widget Accordion può essere usato per un numero qualsiasi di cose, ma la cosa che mi è sempre venuta in mente è una schermata di configurazione... con ciascuna barra che rappresenta un differente insieme di configurazione.

Poi guarderemo all'esempio textlign.py. Non è così "sexy" come l'ultimo, ma è un buon esempio che vi da alcune importanti informazioni che saranno utili successivamente.

Prima di guardare il codice, eseguite il programma.

Quello che dovreste vedere è una etichetta in cima alla finestra, un insieme di nove box rosse con un testo in una griglia 3X3 e quattro pulsanti in fondo della finestra. Come fate clic (toccate) ciascuno dei pulsanti, l'allineamento del testo cambierà. La principale ragione per cui dovreste fare attenzione a questo esempio è come usare e controllare alcuni importanti widget e come cambiare l'allineamento all'interno di essi, che non è completamente intuitivo.

In alto a destra c'è il codice per questo. Lo suddividerò in pezzi; il primo è il codice di importazione (in alto a destra).

Sotto c'è qualcosa di speciale. Hanno creato una classe senza codice. Ne discuterò tra pochi minuti:

```
class BoundedLabel(Label):
  pass
```

Poi viene creata una classe chiamata "Selector" (sotto):

```
from kivy.app import App
from kivy.uix.label import Label
from kivy.uix.gridlayout import GridLayout
from kivy.uix.floatlayout import FloatLayout
from kivy.properties import ObjectProperty
```

```
class Selector(FloatLayout):

app = ObjectProperty(None)
```

Ora la classe Applicazione è stata creata.

Qui viene creata la routine 'select'. Viene creato un widget 'GridLayout' (chiamato griglia), che ha 3 righe e tre colonne. Questa griglia sta per contenere i nove box rosse.

```
for valign in ('bottom',
'middle', 'top'):

for halign in ('left',
'center', 'right'):
```

Qui abbiamo due cicli, uno più interno e uno più esterno.

```
label = BoundedLabel(text='V:
%s\nH: %s' % (valign, halign),

size_hint=(None, None),

halign=halign, valign=valign)
```

Nel codice sopra, viene creata un'istanza del widget 'BoundedLabel', una volta per ciascuna delle nove scatole rosse. Potreste fermarvi qui e dire "Ma aspetta! Non c'è un widget BoundedLabel. C'è solo una istruzione di passaggio dentro". Bene, sì e no. Stiamo creando una istanza di un widget fatto su misura. Come ho detto un poco sopra, ne parleremo un po' di più tra un minuto.

Nel blocco di codice (in alto a destra, prossima pagina) esaminiamo la variabile 'case' che è passata nella routine select.

```
class TextAlignApp(App):

      def select(self, case):

      grid = GridLayout(rows=3, cols=3, spacing=10, size_hint=(None, None),
                  pos_hint={'center_x': .5, 'center_y': .5})
```

Qui la griglia è rimossa, per pulire lo schermo.

```
if self.grid:

self.root.remove_widget(self.gr
id)
```

Il metodo 'bind' imposta le dimensioni e viene aggiunta una griglia all'oggetto radice.

```
grid.bind(minimum_size=grid.set
ter('size'))

self.grid = grid

self.root.add_widget(grid)
```

Ricordate che nell'ultimo esempio ho detto che voi userete quasi sempre una routine di build. Qui è quella per questo esempio. L'oggetto radice è stato creato con un widget FloatLayout. Poi (al centro a destra) chiamiamo la classe Selector per creare un oggetto Selector, poi viene aggiunto all'oggetto radice e inizializziamo la visualizzazione chiamando self.select(0).

Finalmente l'applicazione è pronta per essere eseguita.

```
TextAlignApp().run()
```

Ora, prima di potere andare avanti, dobbiamo chiarire un po' di cose. Primo se voi guardate nella cartella che contiene il file.py noterete un altro file chiamato textalign.kv. Questo è un file speciale che Kivy usa per permettervi di creare i vostri widget e regole. Quando la vostra applicazione Kivy parte, cerca nella stessa directory il file helper.kv. Se c'è, lo carica per primo. Lì è il codice nel file kv.

La prima riga dice a Kivy quale versione minima di Kivy deve essere usata per eseguire questa applicazione.

```
#:kivy 1.0
```

Qui viene creato il widget BoundedLabel. Ciascuna scatola rossa nell'applicazione è una BoundedLabel.

'Color' assegna il colore di sfondo del box a rosso (rgb:1,0,0). Il widget Rectangle crea (avete indovinato) un rettangolo. Quando chiamiamo il widget BoundedLabel nel codice della applicazione, stiamo passando una etichetta come genitore. La dimensione e la posizione (qui nel file .kv) sono assegnate a prescindere dalla dimensione e dalla posizione della etichetta.

Qui (a destra della pagina successiva), vine creato il widget Selector. Questi quattro pulsanti appaiono in fondo alla finestra nello stesso modo in cui le etichette attraversano il margine superiore della finestra.

Notate che l'etichetta che realizza il titolo in cima alla finestra ha una posizione (pos_hint) come top, ha un altezza di 50 pixel e un carattere di dimensione 16. Ciascun pulsante ha un allineamento per il testo al centro. L'istruzione 'on_release' è simile a bind, in modo tale che quando il pulsante viene rilasciato, chiama (in questo caso) root.app.select con un valore a seconda del caso.

Si spera che ora questo inizi ad avere senso. Potete vedere perché Kivy è così potente.

Parliamo in po' dei due widget,

```
if case == 0:
                    label.text_size = (None, None)
          elif case == 1:
                    label.text_size = (label.width, None)
          elif case == 2:
          label.text_size = (None, label.height)
          else:
          label.text_size = label.size
                    grid.add_widget(label)
```

```
     def build(self):
     self.root = FloatLayout()
          self.selector = Selector(app=self)
     self.root.add_widget(self.selector)
     self.grid = None
          self.select(0)
     return self.root
```

```
<BoundedLabel>:
     canvas.before:
          Color:
               rgb: 1, 0, 0
     Rectangle:
          pos: self.pos
               size: self.size
```

GridLayout e FloatLayout, che ho passato prima durante la discussione del codice di esempio.

GridLayout è un widget genitore che usa una descrizione riga e colonna per disporre ogni widget in ciascuna cella. In questo caso è una griglia 3X3 (come il tavoliere per Tic-Tac-Toe (o Naughts e Crosses)).

Quando volete disporre un widget in un GridLayout, usate il metodo add_widget. Qui c'è un problema. Non potete specificare quale controllo va in quale cella della griglia se non nell'ordine con cui li aggiungete. Inoltre, ciascun widget è aggiunto da sinistra a destra, dall'alto in basso. Non potete avere una cella vuota; naturalmente potete imbrogliare, lascio a voi capire come.

Il widget FloatLayout sembra essere solo un contenitore genitore per altri widget figli.

Ho sorvolato su alcuni punti per ora. Il mio intento, questa volta, era semplicemente di eccitarvi sulle possibilità che Kivy ha da offrire. Nella prossima coppia di articoli, continueremo a esplorare che cosa Kivy ha per noi, come usare i vari widget e come creare un APK per pubblicare le nostre applicazioni per Android.

Fino ad allora esplorate ulteriori esempi in Kivy e assicuratevi di andare alle pagine della documentazione per Kivy a http://kivy.org/docs/.

```
<Selector>:
     Label:
     pos_hint: {'top': 1}
          size_hint_y: None
     height: 50
          font_size: 16
     text: 'Demonstration of text valign and halign'
     BoxLayout:
          size_hint_y: None
     height: 50
          ToggleButton:
               halign: 'center'
          group: 'case'
               text: 'label.text_size =\n(None, None)'
               on_release: root.app.select(0)
               state: 'down'
     ToggleButton:
               halign: 'center'
          group: 'case'
               text: 'label.text_size =\n(label.width, None)'
               on_release: root.app.select(1)
     ToggleButton:
               halign: 'center'
               group: 'case'
          text: 'label.text_size =\n(None, label.height)'
               on_release: root.app.select(2)
     ToggleButton:
               halign: 'center'
               group: 'case'
          text: 'label.text_size =\n(label.width, label.height)'
          on_release: root.app.select(3)
```

Greg è il proprietario della RainyDay Solutions, LLC, una società di consulenza in Aurora, Colorado e programma dal 1 972. Ama cucinare, fare escursioni, ascoltare musica e passare il tempo con la sua famiglia. Il suo sito web è www.thedesignatedgeek.net.

Iniziare Python - Parte 36

Scritto da Greg D. Walters

Prima di iniziare, voglio far notare che questo articolo segna tre anni della serie Iniziare a Programmare usando Python. Voglio ringraziare Ronnie e l'intera squadra di Full Circle Magazine per tutto il loro supporto e specialmente voi, i lettori. Non avrei MAI pensato che sarebbe continuato così a lungo.

Voglio inoltre prendere un secondo per notare che ci sono stati alcuni commenti fluttuanti nell'etere secondo i quali, dopo tre anni, la parola "Iniziare" potrebbe essere fuori luogo nel titolo della serie. Dopo tutto, dopo tre anni, siete ancora dei principianti? Bene, a certi livelli, sono d'accordo. Comunque ricevo ancora commenti da lettori che dicono di aver appena scoperto questa serie e Full Circle Magazine e che dunque stanno scorrendo indietro all'inizio della serie. Quindi, questi SONO principianti. Così, a partire dal 37, toglieremo "Iniziare" dal titolo della serie.

Ora passiamo al contenuto vero e proprio di questo articolo… l'approfondimento su Kivy.

Immaginate di suonare la chitarra. Non una immaginaria ma una chitarra vera. Comunque, non siete il miglior suonatore di chitarra e alcuni accordi sono problematici per voi. Per esempio, conoscete gli accordi C, E, G e F (ndt: Do, Mi, Sol e Fa) ma alcuni accordi, come F# minore o C# minore, anche se fattibili, sono difficili da fare per le vostre dita in una canzone veloce. Che cosa fate, specialmente se il concerto è solo tra un paio di settimane e voi DOVETE essere veloci OGGI? La soluzione è usare il capotasto mobile (quella cosa simpatica che si vede qualche volta appesa al manico della chitarra). Questo fa salire la chiave della chitarra e potete usare accordi differenti per armonizzarvi con il resto della banda. Ciò si chiama trasposizione. Alcune volte potete trasporre velocemente nella vostra testa. Altre volte è più facile sedersi davanti a un foglio per farlo se, per esempio, l'accordo è F# minore e, mettendo il capotasto mobile sul tasto 2, potete semplicemente suonare un E minore. Ma questo richiede tempo. Facciamo un applicazione che vi permette di scorrere semplicemente tra le posizioni dei tasti per trovare gli accordi più facili da suonare.

La nostra applicazione sarà abbastanza semplice. Un'etichetta per il titolo, un pulsante con la scala di base come testo, una vista a scorrimento (un meraviglioso widget genitore che contiene altri controlli e vi permette di "lanciare" l'interno del controllo per scorrere) che tiene un numero di pulsanti che hanno riposizionato le scale come testo e un pulsante per uscire. Somiglierà a QUALCOSA come il testo sotto.

Iniziate con un nuovo file python dal nome main.py. Questo sarà importante se e quando deciderete di creare un'applicazione Android da Kivy. Aggiungeremo ora le istruzione di importazione che sono mostrate nella prossima pagina in altro a destra.

Notate la seconda linea, "kivy.require(1.08)". Questo vi permette di essere sicuri di poter usare le ultime e più importanti chicche che Kivy fornisce. Notate anche che stiamo includendo un'uscita di sistema (linea 3). Includeremo alla fine un pulsante per l'uscita.

Qui c'è l'inizio della nostra classe chiamata "Transpose".

```
class Transpose(App):
      def exit(instance):
            sys.exit()
```

```
------------------------------------------------------------------
                        Transposer Ver 0.1

      C  C#/Db  D  D#/Eb   E    F   F#/Gb   G  G#/Ab  A   A#/Bb  B    C

 1| C#/Db  D  D#/Eb  E     F  F#/Gb   G   G#/Ab  A  A#/Bb  B       C   C#/Db
 2|   D  D#/Eb  E     F   F#/Gb  G   G#/Ab  A   A#/Bb  B    C  C#/Db D
------------------------------------------------------------------
```

Ora lavoriamo sulla nostra routine di costruzione (in mezzo a destra). È necessario per ogni applicazione Kivy.

```
import kivy
kivy.require('1.0.8')
from sys import exit
from kivy.app import App
from kivy.core.window import Window
from kivy.uix.button import Button
from kivy.uix.label import Label
from kivy.uix.anchorlayout import AnchorLayout
from kivy.uix.scrollview import ScrollView
from kivy.uix.gridlayout import GridLayout
```

Questo sembra un po' confuso. Sfortunatamente, l'editor non sempre tiene gli spazi corretti perfino con un carattere mono spaziato. L'idea è che la stringa text1 è una semplice scala che inizia con la nota "C". Ciascuna deve essere centrata con 5 spazi. Come il testo mostrato in basso a destra.

La stringa text2 dovrebbe essere la stessa cosa ma ripetuta. Useremo uno spostamento della posizione nella stringa text2 per riempire il pulsante del testo all'interno del widget scrollview.

Ora creiamo l'oggetto root (che è la nostra finestra principale) contenente un widget GridLayout. Se vi ricordate WAY, quando stavamo facendo un altro sviluppo GUI per Glade, c'era un widget gridview. Bene il GridLayout qui è pressappoco lo stesso. In questo caso, abbiamo una griglia che ha una colonna e tre righe. In ciascuna di queste celle create nella griglia possiamo mettere altri widgets. Ricordate, non possiamo definire dove si dispone un widget se non tramite l'ordine in cui lo piazziamo.

```
root =
GridLayout(orientation='vertica
l',spacing=10,cols=1,rows=3)
```

In questo caso, la rappresentazione è come segue ...

```
----------------------------
(0)                title label
----------------------------
(1)                main button
----------------------------
(2)                scrollview
----------------------------
```

La vista a scorrimento contiene elementi multipli, nel nostro caso pulsanti. Successivamente, creiamo l'etichetta che sarà in cima alla nostra applicazione.

```
lbl = Label(text='Transposer
Ver 0.1',
      font_size=20,
      size_hint=(None,None),
      size=(480,20),
      padding=(10,10))
```

Le proprietà impostate dovrebbero essere abbastanza autoesplicative. Le uniche che potrebbero darvi qualche problema potrebbero essere quelle del padding e del size_hint. Il padding è il numero di pixel attorno a un elemento in un riferimento x,y. Preso direttamente dalla documentazione Kivy, size_hint (per X, che è lo stesso per Y) è definito come:

X size_hint. Rappresenta quanto spazio il widget deve usare nella direzione dell'asse X, relativamente alla larghezza del suo genitore. Solo Layout e Window fanno uso di hint: il valore è in percentuale come float da 0. a 1., dove 1 significa l'intera dimensione del genitore, 0.5 rappresenta il 50%.

In questo caso size_hint è impostato a none, il suo valore predefinito è 100% o 1. Questo sarà più importante (e contorto) più avanti.

Ora definiamo il nostro pulsante "principale" (pagina successiva, in alto a destra). Questo è un riferimento statico per la scala.

```
    def build(self):
        #-------------------------------------------------------------
        text1 = "  C  C#/Db  D  D#/Eb  E   F  F#/Gb  G  G#/Ab  A  A#/Bb  B  C"
        text2 = "  C  C#/Db  D  D#/Eb  E   F  F#/Gb  G  G#/Ab  A  A#/Bb  B  C  C#/Db  D
 D#/Eb  E   F  F#/Gb  G  G#/Ab  A  A#/Bb  B    C  C#/Db"
        #-------------------------------------------------------------
```

```
|    |    |    |    |    |    |    |    |    |    |    |    |
12345678901234567890123456789012345678901234567890123456
  C  C#/Db  E   F  F#/Gb  G  G#/Ab  A  A#/Bb  B   C
```

Di nuovo, la maggior parte di questo dovrebbe essere abbastanza chiaro.

Ora aggiungiamo i widget all'oggetto root, che è il widget GridLayout. L'etichetta (lbl) va nella prima cella, il pulsante (btn1) va nella seconda.

```
#--------------------------
root.add_widget(lbl)
root.add_widget(btn1)
#--------------------------
```

Ora arriviamo a del codice più difficile da comprendere. Creiamo un altro oggetto GridLayout e lo chiamiamo "s". Lo colleghiamo all'altezza del prossimo widget che, in questo caso, sarà la ScrollView, NON i pulsanti.

```
s = GridLayout(cols=1, spacing
= 10, size_hint_y = None)
s.bind(minimum_height=s.setter(
'height'))
```

Ora (nel centro a destra) creiamo 20 pulsanti, riempiamone la proprietà testo e quindi aggiungiamoli alla GridLayout.

Adesso creiamo la ScrollView, ne impostiamo la dimensione e la aggiungiamo alla GridLayout root.

```
sv =
ScrollView(size_hint=(None,None
), size=(600,400))

sv.center = Window.center

root.add_widget(sv)
```

Infine aggiungiamo la GridLayout che tiene tutti i nostri pulsanti nella ScrollView e che ritorna l'oggetto root all'applicazione.

```
sv.add_widget(s)

return root
```

Infine, abbiamo la nostra routine "if __name__". Notate che la stiamo configurando per la possibilità di usare l'applicazione come una applicazione android.

```
if __name__ in
('__main__','__android__'):

    Transpose().run()
```

Ora potrete stupirvi del perché ho usato pulsanti invece di etichette per tutti i nostri oggetti testuali. Ciò perché, per impostazione predefinita, le etichette in Kivy non hanno alcun tipo di bordo visibile. Giocheremo con questo nella prossima puntata. Aggiungeremo inoltre un pulsante per l'uscita e qualche altra piccola cosa.

```
    btn1 = Button(text = "  " + text1,size=(680,40),
           size_hint=(None, None),
           halign='left',
           font_name='data/fonts/DroidSansMono.ttf',
           padding=(20,20))
```

```
for i in range(0,19):
    if i <= 12:
        if i < 10:
            t1 = " " + str(i) + "| "
        else:
            t1 = str(i) + "| "
    else:
        t1 = ''
        text2 = ''
    btn = Button(text = t1 + text2[(i*5):(i*5)+65],
          size=(680, 40),
          size_hint=(None,None),
          halign='left',
          font_name='data/fonts/DroidSansMono.ttf')
    s.add_widget(btn)
#------------------------------------------------------------
```

Il codice sorgente può essere trovato su PasteBin presso http://pastebin.com/hsicnyt1.

Fino alla prossima volta, divertitevi e vi ringrazio per avermi sopportato per tre anni!

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

Programmare in Python - Parte 37

Scritto da Greg D. Walters

Questo mese finiremo il programma traspositore che abbiamo scritto in Kivy. Spero che abbiate salvato il codice dall'ultima volta perchè costruiremo sopra questo. Se non è così, allora prendete il codice dal numero 64 di FCM.

Incominciamo ricapitolando quello che abbiamo fatto la volta scorsa. Abbiamo creato una applicazione che permette a un chitarrista di trasporre velocemente da una chiave all'altra. Il fine ultimo è quello di essere in grado di eseguire questa applicazione non solo in Linux o in una scatola Windows ma anche altrettanto bene sui dispositivi Android. Prendo la mia sul mio tablet ogni volta che vado con la band per fare pratica. Stavo per trattare con la preparazione del nostro progetto per android, ma alcune cose sono cambiate nel metodo per farlo, così lavoreremo su questo la prossima settimana.

L'applicazione, come la abbiamo lasciata l'ultima volta sembra come quella mostrata sotto a sinistra.

Quando abbiamo fatto, dovrebbe sembrare come lo schermo sotto a destra.

La prima cosa che noterete è che ci sono delle etichette blu piuttosto che quelle grigie noiose. Il prossimo è che ci sono tre pulsanti. Infine le etichette scorrevoli sono più vicine all'intera larghezza della finestra. Oltre a questo è quasi dal tutto (visualmente) lo stesso.

Uno dei pulsanti è un pulsante "about" che vi farà saltar fuori delle semplici informazioni, ma spiega come fare un semplice popup. Uno dei pulsanti è un pulsante per uscire. L'altro pulsante inverte l'etichetta di testo per rendere facile la trasposizione da pianoforte a chitarra o da chitarra a pianoforte.

```
#:kivy 1.0
#:import kivy kivy

<BoundedLabel>:
      canvas.before:
      Color:
            rgb: 0, 0, 1
      Rectangle:
            pos: self.pos
            size: self.size
```

Incominciamo creando un file .kv (sopra a destra). Questo è quello che ci

indice ^

da le etichette colorate. E' un file veramente semplice.

Le prime due linee sono obbligatorie. Semplicemente dicono quale versione di Kivy aspettarsi. Poi creiamo un nuovo tipo di etichetta chiamato "BoundedLevel". Il colore è assegnato con valori RGB (tra 0 e 1, che può essere considerato come il 100%) e come potete vedere il valore blu è assegnato al 100 percento. Creiamo inoltre un rettangolo che è la vera etichetta: salvate questo come "transpose.kv". Dovete utilizzare il nome della classe che utilizzeremo.

Ora che avete completato questo, aggiungete le seguenti linee giusto prima della classe transpose al file sorgente dall'ultima volta.

```
class BoundedLabel(Label):

    pass
```

Per fare in modo che tutto funzioni, tutto ciò di cui abbiamo bisogno è una definizione. Prima di andare avanti, aggiungete la seguente linea alla sezione import:

```
from kivy.uix.popup import Popup
```

Questo ci permette di creare il popup dopo. Ora nella classe Transpose, proprio dentro alla routine della definizione della costruzione, piazzate il codice sopra a destra.

```
def LoadLabels(w):
    if w == 0:
            tex0 = self.text1
            tex1 = self.text2
        else:
            tex0 = self.text3
            tex1 = self.text4
        for i in range(0,22):
            if i <= 12:
               if i < 10:
                  t1 = " " + str(i) + "| "
               else:
                  t1 = str(i) + "| "
                  t = tex1
            else:
                t1 = ''
                t = ''
            l = BoundedLabel(text=t1+t[(i*6):(i*6)+78], size=(780, 35),
                size_hint=(None,None),halign='left',
                font_name='data/fonts/DroidSansMono.ttf')
            s.add_widget(l)
```

La routine LoadLabels ci darà le etichette colorate (BoundedLabel) e la capacità di invertire. Avete visto la maggior parte la volta scorsa. Passiamo un valore al parametro "w" per determinare quale testo deve essere mostrato. La linea l=BoundedLabel è molto simile alla stessa linea dell'ultima volta, con l'eccezione che, questa volta, stiamo usando un widget BoundedLabel invece di un widget Button. La Loadlabels sarà principalmente

```
def Swap(instance):
    if self.whichway == 0:
        self.whichway = 1
        btnWhich.text = "Guitar --> Piano"
        btn1.text = "    " + self.text3
        s.clear_widgets()
        LoadLabels(1)
    else:
        self.whichway = 0
        btnWhich.text = "Piano --> Guitar"
        btn1.text = "    " + self.text1
        s.clear_widgets()
        LoadLabels(0)
```

```
self.whichway=0
self.text1 = "  C  |  B  |A#/Bb|  A  |G#/Ab|  G  |F#/Gb|  F  |  E  |D#/Eb|  D  |C#/Db|  C  |"
self.text2 = "  C  |  B  |A#/Bb|  A  |G#/Ab|  G  |F#/Gb|  F  |  E  |D#/Eb|  D  |C#/Db|  C  |  B  |A#/Bb|  A  |G#/Ab|  G  |F#/Gb|  F  |  E  |D#/Ab|  D  |C#/Db|  C  |"
self.text3 = "  C  |C#/Db|  D  |D#/Eb|  E  |  F  |F#/Gb|  G  |G#/Ab|  A  |A#/Bb|  B  |  C  |"
self.text4 = "  C  |C#/Db|  D  |D#/Eb|  E  |  F  |F#/Gb|  G  |G#/Ab|  A  |A#/Bb|  B  |  C  |C#/Db|  D  |D#/Eb|  E  |  F  |F#/Gb|  G  |G#/Ab|  A  |A#/Bb|  B  |  C  |C#/Db|"
```

chiamata dalla prossima routine, Swap. Piazzate questo codice (mostrato a destra) sotto LoadLabels.

Voi potete vedere che questa routine è autoesplicativa. Usiamo una variabile (self.whichway) per determinare "in quale modo" le etichette sono visualizzate … dalla Chitarra al Pianoforte o dal Pianoforte alla Chitarra.

Assicuratevi di salvare il vostro lavoro a questo punto dato che stiamo per fare molti cambiamenti da qui in avanti.

Sostituite le linee che definiscono text1 e text2 con le linee mostrate sopra.

Assegniamo a zero self.whichway che sarà il nostro valore di default per la procedura di inversione. Quindi definiamo quattro stringhe invece delle due che avevamo l'ultima volta. Potete notare che le stringhe text3 e text4 sono semplici inversioni di text1 e di text2.

Ora aggiustiamo la linea contenente la definizioni di base. Cambiatela da

```
root =
GridLayout(orientation='vertical
', spacing=10, cols=1,rows=3)
```

a

```
root =
GridLayout(orientation='vertical
', spacing=6, cols=1, rows=4,
row_default_height=40)
```

Abbiamo cambiato la spaziatura da 10 a 6 e assegnato l'altezza predefinita della riga a 40 pixel. Cambiate il testo per l'etichetta (successiva linea) a "text='Transposer Ver 0.8.0'". Tutto il resto resta lo stesso su questa linea.

Ora cambiate la linea della definizione dei pulsanti da …

```
btn1 = Button(text = "  " +
text1,size=(680,40),
size_hint=(None,None),
halign='left',
font_name='data/fonts/DroidSansM
ono.ttf', padding=(20,20))
```

a:

```
btn1 = Button(text = "  " +
self.text1,size=(780,20),
size_hint=(None, None),
halign='left',
font_name='data/fonts/DroidSansM
ono.ttf', padding=(20,2),
background_color=[0.39,0.07,.92,
1])
```

Notate che ho cambiato la formattazione della prima definizione per chiarezza. I grandi cambiamenti sono il cambiamento delle dimensioni da 680,40 a 780, 20 e il colore di sfondo per il pulsante. Ricordate che possiamo cambiare il colore di sfondo per i pulsanti ma non per le etichette.

Poi, definiamo tre widget AnchorLayout per i tre pulsanti che aggiungeremo più tardi. Li ho nominate al0 (AnchorLayout0) , al1 e al2. Abbiamo anche aggiunto il codice per il Popup About e definito i nostri pulsanti insieme con le definizioni di collegamento

Questo è mostrato sulla prossima pagina in alto a sinistra.

Trovate la linea "s = GridLayout" e cambiate la spaziatura da 10 a 4. Poi aggiungete la seguente linea dopo la linea s.bind (a destra prima del ciclo for):

```
LoadLabels(0)
```

Questo chiama la routine LoadLabels con il nostro "which" di default a 0.

Poi, commentate per intero il codice del ciclo. Questo inizia con "for i in range(0,19):" e finisce con "s.add_widget(btn)". Noi non abbiamo bisogno di questo, poiché la routine LoadLabels fa questo per noi.

Ora salvate il vostro codice e provate a eseguirlo. Dovreste vedere un profondo pulsante viola in cima e le nostre simpatiche BoundLabels blu. In più noterete che le BoundLabels nella

```
al0 = AnchorLayout()
al1 = AnchorLayout()
al2 = AnchorLayout()
popup = Popup(title='About Transposer',
        content=Label(text='Written by G.D. Walters'),
        size_hint=(None,None),size=(400,400))
btnWhich = Button(text = "Piano --> Guitar",
           size=(180,40),size_hint=(None,None))
btnWhich.bind(on_release=Swap)
btnAbout = Button(text="About",size=(180,40),
           size_hint=(None,None))
btnAbout.bind(on_release=ShowAbout)
btnExit = Button(text="Exit", size=(180,40),
          size_hint=(None,None))
btnExit.bind(on_release=exit)
```

finestra di scorrimento sono più vicine, che rende molto più facile da leggere.

Abbiamo quasi finito con il nostro codice ma abbiamo ancora un po' di cose da fare. Dopo la linea "sv = ScrollView" aggiungete la seguente linea …

```
sv.size = (720, 320)
```

Questo assegna la dimensione del widget ScrollView a 720 per 320 che lo fa più largo nella finestra principale. Ora prima della linea "return root" aggiungete il codice mostrato in alto a destra.

Qui abbiamo aggiunto i tre pulsanti ai widget AnchorLayout. Create un GridLayout per contenere gli Anchor Layout, e quindi finalmente aggiungete gli AnchorLayout alla GridLayout.

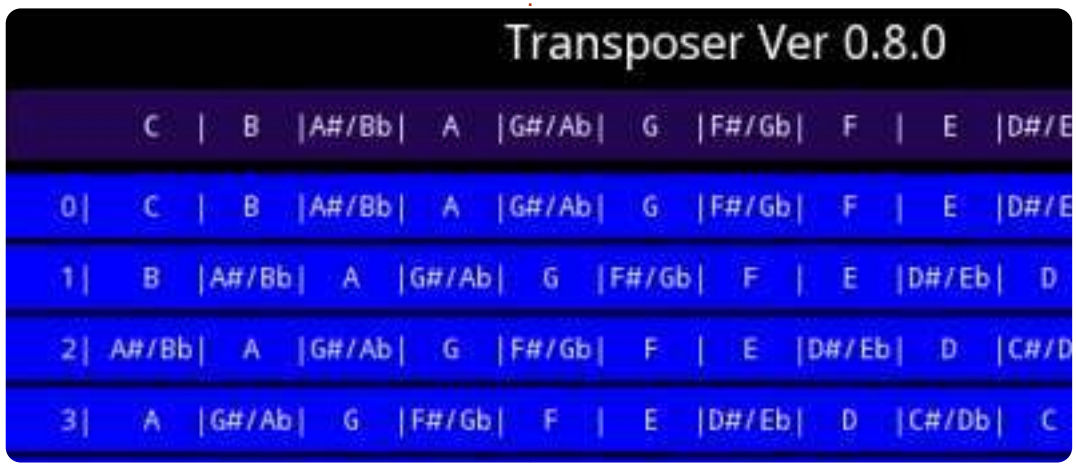
Tornate indietro giusto sotto alla routine "def Swap" e aggiungete i seguenti …

```
def ShowAbout(instance):
popup.open()
```

Questo è tutto. Salvate ed eseguite il codice. Se cliccate sul pulsante About, vedrete il semplice popup. Premete in qualsiasi punto al di fuori del popup per fare in modo che scompaia. Ora il vostro codice è scritto. Potete trovare tutto il codice scritto a http://pastebin.com/GftmjENs

Ora dobbiamo creare il nostro pacchetto Android … ma dovete aspettare la prossima volta.

```
        al0.add_widget(btnWhich)
        al1.add_widget(btnExit)
        al2.add_widget(btnAbout)
        bgl = GridLayout(orientation='vertical',
              spacing=6, cols=3,rows=1,
              row_default_height=40)
        bgl.add_widget(al0)
        bgl.add_widget(al1)
        bgl.add_widget(al2)
```



Se volete configurare e provare a impacchettare per Android prima del prossimo mese dovete andare su http://kivy.org/docs/guide/packaging-android.html per la documentazione su questo: assicuratevi di seguire la documentazione con cura.

Arrivederci al prossimo mese

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

Programmare in Python: Parte 38

Scritto da Greg Walters

Come ho promesso nella parte 37, prenderemo la app del traspositore che abbiamo creato, e creeremo un APK per installarlo sul vostro dispositivo Android.

Prima di cominciare, assicuriamoci di avere tutto pronto. Per prima cosa abbiamo bisogno dei due file che abbiamo creato l'ultima volta in una cartella a cui voi potete accedere facilmente. Chiamatela "transposer". Createla nella vostra home directory, quindi copiate i due file (transpose.kv e transpose.py) in questa cartella. Ora rinominate transpose.py in main.py. Questa parte è importante.

Poi abbiamo bisogno di fare riferimento alle istruzioni di packaging di Kivy in un browser web. Il collegamento è http://kivy.org/docs/guide/packaging-android.html. Useremo questo per i prossimi passi, ma non esattamente come intendevano le persone di Kivy. Dovreste avere l'SDK android dalle lezioni precedenti. Idealmente seguirete le istruzioni e vi procurerete tutto il software che è elencato qui, ma per i nostri propositi, potete semplicemente seguire qui. Dovete scaricare il software python-for-android. Aprite la finestra del terminale e digitate i seguenti comandi...

```
./build.py --dir <path to your app>
           --name "<title>"
           --package <org.of.your.app>
           --version <human version>
           --icon <path to an icon to use>
           --orientation <landscape|portrait>
           --permission <android permission like VIBRATE> (multiple allowed)
           <debug|release> <installd|installr|...>
```

```
git clone
git://github.com/kivy/python-
for-android
```

Questo scaricherà e configurerà il software di cui abbiamo bisogno per continuare. Ora nella finestra del terminale, cambiate la vostra directory alla cartella python-for-android/dist/default folder.

Ora troverete un file chiamato build.py. Questo è quello che farà tutto il lavoro per noi. Ora arriva la magia.

Il programma build.py prenderà vari argomenti da linea di comando e creerà l'APK per voi. Sotto trovate la sintassi usata per build.py presa direttamente dalla documentazione Kivy.

Per il nostro scopo, useremo il seguente comando (il carattere "\" è un carattere per indicare la continuazione di linea):

```
./build.py --dir ~/transposer
--package
org.RainyDay.transposer \
 --name "RainyDay Transposer"
--version 1.0.0 debug
```

Diamo un'occhiata ai pezzi del comando...

./build.py – questa è l'applicazione
--dir ~/transposer – questa è la directory dove si trova il codice della nostra applicazione.
--package org.RainyDay.transposer – questo è il nome del pacchetto
--name "RainyDay Transposer" – questo è il nome della applicazione che sarà mostrato nel drawer (o cassetto) delle applicazioni.
--version 1.0.0 – la versione della nostra applicazione
debug – questo è il livello della release (debug o release)

Una volta che lo avete eseguite, assumendo che tutto funzioni come ci si aspetta, dovreste avere un certo numero di file nella cartella /bin. Quello che state cercando si chiama "RainyDayTransposer-1.0.0-debug.apk". Potete copiarlo nel vostro dispositivo android usando il vostro file manager preferito, e installarlo come qualsiasi altra applicazione dai vari app stores.

Questo è tutto quello che ho potuto scrivere questo mese.