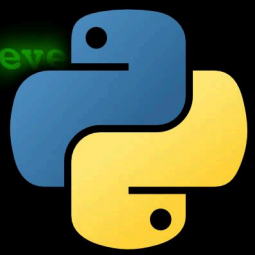
full circle

LA RIVISTA INDIPENDENTE PER LA COMUNITÀ LINUX UBUNTU

EDIZIONE SPECIALE SERIE PROGRAMMAZIONE

EDIZIONE SPECIALE
SERIE PROGRAMMAZIONE



python™

PROGRAMMARE IN PYTHON VOLUME 3

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 17-21**, pubblicata nei numeri 43-47: niente di speciale, giusto quello che abbiamo già pubblicato.

Vi chiediamo, però, di badare alla 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
Podcaster: Robin Catling (aka RobinCatling)
podcast@fullcirclemagazine.org
Manager delle comunicazioni: Robert Clipsham (aka: mrmonday) -
mrmonday@fullcirclemagazine.org

Programmare in Python - Parte 17

Scritto da Greg Walters

Mentre stavo concludendo l'ultimo articolo della nostra serie, ho ricevuto una mail riguardante una gara di programmazione. Anche se non abbiamo tempo di occuparcene, diversi siti propongono gare del genere durante l'anno. Le informazioni sono reperibili all'indirizzo http://www.freiesmagazin.de/third_programming_contest, se siete interessati. Tutto questo mi ha fatto pensare che non abbiamo mai parlato di una vera programmazione Client/Server. Quindi, con questo obiettivo in mente ci addentriamo nell'argomento e vediamo dove ci porta.

Cos'è un'applicazione Client/Server? Semplicemente, ogniqualvolta usate un programma (o anche una interfaccia web) che accede ai dati di un'altra applicazione o computer allora state usando un sistema client/server. Consideriamo un esempio che ci è familiare. Ricordate quando abbiamo realizzato il programma delle ricette? Era un esempio MOLTO semplice (e nemmeno tanto bello) di applicazione client/server. Il database SQLITE era il server, l'applicazione che abbiamo scritto era il client. Un esempio più calzante è il seguente. Consideriamo un database su un computer ubicato diversi piani lontano dal vostro ufficio. Contiene informazioni sull'inventario del negozio in cui lavorate. Usate un registratore di cassa (uno di 10) all'interno del negozio. Ciascuno di questi registratori rappresenta un client e il database posizionato da qualche parte è il server.

Anche se non creeremo quel genere di sistema, possiamo impararne i concetti base.

La prima cosa a cui bisogna pensare è l'ubicazione del nostro server. Molte persone hanno solo un computer nella propria casa. Altre ne possiedono 7 o 8.

Per utilizzare il sistema client/server, bisogna connettersi dalla macchina client a quella server. Lo facciamo attraverso quello che chiamiamo pipe o socket. Se da piccoli avete creato un telefono con filo e lattine allora avete un'idea di cosa sto parlando. Altrimenti, lasciate che ve lo descriva. Prima di tutto, dovevate ottenere da vostra madre un paio di lattine di fagioli in scatola o qualcosa di simile. Quindi, dopo averle pulite per bene le si portava in garage. Si usava un chiodo sottile e un martello per creare un forellino sul fondo di ciascuna. Quindi bisognava procurarsi 4-5 metri di filo (ancora dalla amata mamma), far passare l'estremità attraverso ciascuna lattina e realizzare un nodo per evitare che l'estremità fuoriuscisse. Quindi ci si incontrava con il miglior amico e, tenendo ben teso il filo, si parlava nella lattina mentre il nostro amico teneva l'altra sull'orecchio. Le vibrazioni del fondo si propagano lungo il filo teso provocando la vibrazione del fondo dell'altra lattina. Ovviamente avreste potuto sentire anche senza, ma il punto non è questo. Era divertente. Il socket è più o meno la stessa cosa. Il client ha una connessione diretta (pensate al filo) al server. Se molti client sono connessi al server, ogni client utilizzerà una lattina propria e il povero server deve avere lo stesso numero di lattine tutte saldamente collegate a quelle corrispondenti dei client.

Creiamo un client/server semplice. Inizieremo dal server. In pseudo codice, ecco cosa accade.

Crea un socket
Recupera il nome del server
Seleziona una porta
Associa il socket all'indirizzo e alla porta
Resta in attesa della connessione
Se connesso...
** Accetta la connessione**
** Stampa che abbiamo la connessione**
** Chiudi la connessione**

Il vero codice per il nostro server si trova nella pagina seguente, in basso a sinistra.

Allora, abbiamo creato il socket, recuperato il nome della macchina host su cui il server è in esecuzione, associato il socket alla porta e

iniziato ad ascoltare. Quando riceviamo una richiesta di connessione, l'accettiamo, stampiamo il fatto che siamo connessi, inviamo "Hello and Goodbye" e chiudiamo il socket.

Ora abbiamo bisogno del client per rendere il tutto funzionante (mostrato in basso a destra).

Il codice è simile a quello del server ma, in questo caso, ci connettiamo, stampiamo quanto ricevuto e chiudiamo il socket.

L'output dei programmi è molto prevedibile. Sul lato server otteniamo...

```
My hostname is earth

I'm now connected to
('127.0.1.1', 45879)
```

e su quello client...

```
Hello and Goodbye
```

Quindi tutto molto semplice. Ora rendiamo il tutto un po' più realistico. Creeremo un server che effettivamente fa qualcosa. Il codice della seconda versione lo trovate qui: http://fullcirclemagazine.pastebin.com/Az8vNUv7

Analizziamolo a blocchi. Dopo la sezione degli import, impostiamo alcune variabili. BUFSIZ contiene le dimensioni del buffer che useremo per conservare l'informazione ricevuta dal client. Impostiamo anche la porta su cui ascoltare e una lista con l'host e il numero della porta.

```
#!/usr/bin/env python
#server1.py
import socket
soc = socket.socket()
hostname = socket.gethostname()
print "My hostname is ", hostname
port = 21000
soc.bind((hostname,port))
soc.listen(5)
while True:
     con,address = soc.accept()
     print "I'm now connected to ",address
     con.send("Hello and Goodbye")
     con.close()
```

Quindi creiamo la classe ServCmd. Nella funzione __init__ inseriamo il socket e vi associamo l'interfaccia. Nella funzione run iniziamo ad ascoltare e aspettiamo il comando dal client.

```
#!/usr/bin/env python
# client2.py

from socket import *
from time import time
from time import sleep
import sys
BUFSIZE = 4096

class CmdLine:
    def __init__(self,host):
        self.HOST = host
        self.PORT = 29876
        self.ADDR = (self.HOST,self.PORT)
        self.sock = None

    def makeConnection(self):
        self.sock = socket( AF_INET,SOCK_STREAM)
        self.sock.connect(self.ADDR)

    def sendCmd(self, cmd):
        self.sock.send(cmd)

    def getResults(self):
        data = self.sock.recv(BUFSIZE)
        print data

if __name__ == '__main__':
    conn = CmdLine('localhost')
    conn.makeConnection()
    conn.sendCmd('ls -al')
    conn.getResults()
    conn.sendCmd('BYE')
```

```
#!/usr/bin/python
# client1.py
#====================
import socket

soc = socket.socket()
hostname = socket.gethostname()
port = 21000

soc.connect((hostname, port))
print soc.recv(1024)
soc.close
```

Quando ne riceviamo uno, usiamo la routine os.popen(). Questa si occupa di creare un terminale ed eseguire il comando.

Quindi il client (in alto a destra), la cui creazione è un compito semplice.

Saltiamo tutto eccetto il comando di invio, dato che ora avete abbastanza informazioni per comprenderlo da soli. La riga conn.sendCmd() (riga 31) invia una semplice richiesta ls -al. Ecco come appare la risposta nel mio caso. La vostra potrebbe essere differente.

Server

```
python server2.py
...listening
...connected:
('127.0.0.1',42198)
Command received - ls -al
Command received - BYE
...listening
```

Client

```
python client2a.py
total 72
drwxr-xr-x 2 greg greg 4096
2010-11-08 05:49 .
drwxr-xr-x 5 greg greg 4096
2010-11-04 06:29 ..
-rw-r--r-- 1 greg greg 751
2010-11-08 05:31 client2a.py
-rw-r--r-- 1 greg greg 760
2010-11-08 05:28 client2a.py~
-rw-r--r-- 1 greg greg 737
2010-11-08 05:25 client2.py
-rw-r--r-- 1 greg greg 733
2010-11-08 04:37 client2.py~
-rw-r--r-- 1 greg greg 1595
2010-11-08 05:30 client2.pyc
-rw-r--r-- 1 greg greg 449
2010-11-07 07:38 ping2.py
-rw-r--r-- 1 greg greg 466
2010-11-07 10:01
python_client1.py
-rw-r--r-- 1 greg greg 466
2010-11-07 10:01
python_client1.py~
-rw-r--r-- 1 greg greg 691
2010-11-07 09:51
python_server1.py
-rw-r--r-- 1 greg greg 666
2010-11-06 06:57
python_server1.py~
-rw-r--r-- 1 greg greg 445
2010-11-04 06:29 re-test1.py
-rw-r--r-- 1 greg greg 1318
2010-11-08 05:49 server2a.py
-rw-r--r-- 1 greg greg 1302
2010-11-08 05:30 server2a.py~
-rw-r--r-- 1 greg greg 1268
2010-11-06 08:02 server2.py
-rw-r--r-- 1 greg greg 1445
2010-11-06 07:50 server2.py~
-rw-r--r-- 1 greg greg 2279
2010-11-08 05:30 server2.pyc
```

Possiamo anche connetterci da un'altra macchina senza cambiare nulla, con la sola eccezione di conn = CmdLine('localhost') (riga 29) nel programma client. In questo caso, cambiate 'localhost' con l'indirizzo IP della macchina che esegue il server. Per la mia configurazione casalinga uso la seguente riga:

```
conn =
CmdLine('192.168.2.12')
```

Siamo così in grado di inviare informazioni avanti e indietro da una macchina (o terminale) all'altra.

La prossima volta renderemo più robusta la nostra applicazione.

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

Cercasi idee e scrittori

Su LaunchPad abbiamo creato le pagine del progetto e della squadra Full Circle. L'idea è quella che i non-scrittori possono collegarsi alla pagina, fare clic su "Answers" in alto e lasciare idee per articoli, **ma vi prego siate specifici!** Non inserite solo "articolo sui server" ma indicate anche cosa il server dovrebbe fare!

I lettori che volessero scrivere un articolo ma sono a corto di idee, possono registrarsi alla pagina del gruppo Full Circle quindi auto-assegnarsi gli articoli proposti e iniziare a scrivere! Chiediamo che **se non è possibile scrivere l'articolo nel giro di alcune settimane (un mese circa) la richiesta venga riaperta** per permettere a qualcun'altro di adottarla.

Pagina del progetto, **per le idee**:
https://launchpad.net/fullcircle
Gruppo degli **scrittori**:
https://launchpad.net/~fullcircle

Programmare in Python - Parte 18

Scritto da Greg Walters

L'ultima volta abbiamo creato un sistema client/server molto semplice. Questa volta lo espanderemo un po'. Il server sarà una tavola e un validatore del gioco del Tris (o cerchi e croci). La parte client fungerà da input/output.

Inizieremo usando lo stesso codice del server dell'ultima volta e lo modificheremo strada facendo. Se non lo avete ancora salvato, andate su http://fullcirclemagazine.pastebin.com/UhquVK4N, recuperate il codice e proseguite. Il primo cambiamento riguarda la routine __init__ dove inizializzeremo due nuove variabili, self.player e self.gameboard. Gameboard è un semplice elenco di liste o un array base. Possiamo accedervi come segue (maggiormente intuibile di una lista piatta). Questa lista conterrà i nostri dati. Ogni cella accetterà tre possibili valori. "-" indica una cella vuota, "X" una cella occupata dal giocatore 1 e "O" una occupata dal giocatore 2. La griglia assomiglia alla seguente quando la rappresentiamo in due dimensioni:

```
[0][0] | [0][1] | [0][2]
[1][0] | [1][1] | [1][2]
[2][0] | [2][1] | [2][2]
```

Così, partendo dal codice del server del mese passato, nella funzione __init__ inseriamo le seguenti righe:

```
# The next three lines are
new...
    self.player = 1
    self.gameboard = [['-','-
','-'],['-','-','-'],['-','-
','-']]

    self.run()
```

Le routine run, listen, e servCmd non sono cambiate, quindi ci concentriamo sulle modifiche a procCmd.

Nell'articolo dell'ultima volta, il server attendeva un comando dal client e lo inviava alla funzione os.popen. Questa volta analizzeremo il comando ricevuto. In questo caso resteremo in attesa di tre comandi differenti. Sono "Start", "Move", e "GOODBYE". Quando riceviamo il comando "Start", il server dovrà inizializzare la scacchiera con degli "0" e quindi inviare la "rappresentazione" della tavola al client.

Il comando "Move" è un comando composto in quanto contiene il comando e la posizione nella quale il giocatore vuole muoversi. Per esempio, "Move A3". Analizzando il comando ricaveremo tre informazioni, lo stesso comando "move", la riga e la colonna. Per finire, il comando "GOODBYE" resetta, semplicemente, la tavola da gioco per un'altra partita.

Allora, riceviamo il comando dal client nella funzione procCmd. Quindi lo analizziamo per vedere cosa fare. All'interno della routine procCmd, portiamoci alla 5a riga e rimuoviamo tutto il codice che segue la riga contenente "if self.processingloop:". Ora imposteremo i comandi così come li abbiamo definiti. Ecco il codice per il comando Start:

```
if self.processingloop:
    if cmd == 'Start':
        self.InitGameBoard()
        self.PrintGameBoard(1)
```

Proseguiamo con la porzione della routine che riguarda Move (mostrata in basso). Prima controlliamo i primi quattro caratteri del comando passato per vedere se corrispondono a "Move". In caso affermativo estraiamo la parte restante della stringa a partire dalla posizione 5 (dato che l'indice base è 0) e l'assegniamo alla variabile chiamata position. Quindi controlliamo se il primo carattere è una "A", "B", o "C". Rappresenta la riga inviata dal client. Quindi recuperiamo l'intero del carattere successivo e avremo la nostra colonna:

```
if cmd[:4] == 'Move':
print "MOVE COMMAND"
position = cmd[5:]
if position[0] == 'A':
    row = 0
elif position[0] == 'B':
    row = 1

elif position[0] == 'C':
    row = 2
else:
    self.cli.send('Invalid
position')
    return
col = int(position[1])-1
```

A seguire, facciamo un controllo rapido per assicurarci che la posizione della riga sia tra quelle possibili:

```
if row < 0 or row > 2:
    self.cli.send('Invalid
position')
    return
```

Per finire, verifichiamo che la posizione sia vuota ("-") e, se il giocatore corrente è il numero 1, mettiamo una "X", altrimenti "O". Quindi chiamiamo la funzione PrintGameBoard con "0" come parametro:

```
if self.gameboard[row][col]
== '-':
    if self.player == 1:

self.gameboard[row][col] =
"X"
    else:

self.gameboard[row][col] =
"O"

self.PrintGameBoard(0)
```

Questo conclude le modifiche alla funzione procCmd. Ora tocca alla funzione "inizializza la scacchiera di gioco". Tutto quello che fa è impostare ciascuna posizione a "-", che la logica di move usa per verificare che uno spazio sia vuoto:

```
def InitGameBoard(self):
    self.gameboard = [['-','-
','-'],['-','-','-'],['-','-
','-']]
```

La routine PrintGameBoard (sotto) stampa la scacchiera, chiama la funzione checkwin e imposta il numero del giocatore. Creiamo una stringa molto grande da inviare al client così da dover inserire la funzione d'ascolto una volta per mossa. Il parametro firsttime è incluso per inviare il pritty print del tavolo da gioco quando il client si connette per la prima volta o resetta il gioco:

Quindi, controlliamo se il parametro è impostato a 0 o 1 (sotto). Solo se firsttime è 0 controlliamo se il giocatore attuale ha vinto e, se è così, aggiungiamo il testo "Player X WINS!" alla stringa in uscita. Se il giocatore corrente non ha vinto allora inseriamo il testo "Enter move...". Per finire inviamo la stringa al client con la funzione cli.send:

```
if firsttime == 0:
            if self.player == 1:
                ret = self.checkwin("X")
            else:
                ret = self.checkwin("O")
            if ret == True:
                if self.player == 1:
                    outp += "Player 1 WINS!"
                else:
                    outp += "Player 2 WINS!"
            else:
                if self.player == 1:
                    self.player = 2
                else:
                    self.player = 1
                outp += ('Enter move for player %s' %
self.player)
        self.cli.send(outp)
```

Finalmente, nella pagina successiva, ecco la routine usata dal server per controllare l'evento vittoria. Abbiamo già impostato per il giocatore una "X" o "O", quindi iniziamo usando un semplice ciclo for. Se ci imbattiamo in una vittoria, ritorniamo True. La variabile "C" del ciclo for rappresenta ciascuna riga nella nostra lista di liste. Prima controlleremo ogni riga per una vittoria in orizzontale:

```
def PrintGameBoard(self,firsttime):
    #Print the header row
    outp = ('   1   2   3') + chr(13) + chr(10)
    outp += (" A {0} | {1} | {2}".format(self.gameboard[0][0],self.gameboard[0][1],self.gameboard[0][2])) + chr(13)+chr(10)
    outp += ('  ------------')+ chr(13)+chr(10)
    outp += (" B {0} | {1} | {2}".format(self.gameboard[1][0],self.gameboard[1][1],self.gameboard[1][2]))+ chr(13)+chr(10)
    outp += ('  ------------')+ chr(13)+chr(10)
    outp += (" C {0} | {1} | {2}".format(self.gameboard[2][0],self.gameboard[2][1],self.gameboard[2][2]))+ chr(13)+chr(10)
    outp += ('  ------------')+ chr(13)+chr(10)
```

```
First, we will check each Row for a horizontal win:
   def checkwin(self,player):
        #loop through rows and columns
        for c in range(0,3):
        #check for horizontal line
            if self.gameboard[c][0] == player and
self.gameboard[c][1] == player and self.gameboard[c][2] ==
player:
              print "*********\n\n%s wins\n\n*********" %
player
              playerwin = True
              return playerwin

Next, we check each Column for a win:
            #check for vertical line
            elif self.gameboard[0][c] == player and
self.gameboard[1][c] == player and self.gameboard[2][c] ==
player:
              print "** %s wins **" % player
              playerwin = True
              return playerwin

Now we check for the diagonal win from left to right...
            #check for diagonal win (left to right)
            elif self.gameboard[0][0] == player and
self.gameboard[1][1] == player and self.gameboard[2][2] ==
player:
              print "** %s wins **" % player
              playerwin = True
              return playerwin

Then from right to left...
            #check for diagonal win (right to left)
            elif self.gameboard[0][2] == player and
self.gameboard[1][1] == player and self.gameboard[2][0] ==
player:
              print "** %s wins **" % player
              playerwin = True
              return playerwin

Finally, if there is no win, we return false:
        else:
            playerwin = False
            return playerwin
```

Il Client

Ancora una volta, iniziamo dalla semplice routine creata l'ultima volta. I cambiamenti iniziano subito dopo la chiamata a conn.makeConnection. Inviamo i comandi Start, diversi Move e, per finire, Goodbye. La cosa più importante da ricordare qui è che dopo aver inviato un comando dovete attendere la risposta prima di inviarne un altro. Immaginatela come una conversazione educata. Inviate la vostra richiesta, attendete la risposta, inviatene un'altra, attendete nuovamente, e così via. In questo esempio usiamo raw_input semplicemente perché possiate vedere cosa avviene:

```
if __name__ == '__main__':
    conn =
CmdLine('localhost')
    conn.makeConnection()
    conn.sendCmd('Start')
    conn.getResults()
    conn.sendCmd('Move A3')
    conn.getResults()
    r = raw_input("Press
Enter")
    conn.sendCmd('Move B2')
    conn.getResults()
    r = raw_input("Press
Enter")
```

Continuate con la tripletta sendCmd, getResults, raw_input con ciascuno dei seguenti comandi (dovreste già avere il codice per le mosse A3 e B2), C1, A1, C3, B3, C2 e quindi terminate con il comando GOODBYE.

Ulteriori modifiche

Ecco i "compiti per casa". Nel client, rimuovete i comandi preparati per le mosse e usate raw_input() per richiedere e catturare le mosse dal/dai giocatore/i nella forma "A3" o "B2" e quindi farli precedere da "Move" prima di inviare l'intero comando al server.

La prossima volta modificheremo il nostro server per giocare come avversario.

Il codice del Client e del Server può essere trovato a questi indirizzi: http://fullcirclemagazine.pastebin.com/UhquVK4N o http://thedesignatedgeek.com.

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

Programmare in Python - Parte 19

Questa volta lavoreremo per terminare il nostro programma 'Tris'. Tuttavia, diversamente dalla maggior parte dei miei altri articoli, non fornirò il codice. Lo farete voi. Comunque vi darò le regole. Dopo 18 mesi, avete gli strumenti e le conoscenze per terminare questo progetto. Ne sono sicuro.

Per prima cosa diamo uno sguardo alla logica del gioco del Tris. Vedremo il suo pseudo-codice. Diamo prima un'occhiata al tavolo da gioco. È disposto così...

```
Spigolo|  Lato   | Spigolo
-------+--------+-------
 Lato  | Centro | Spigolo
-------+--------+-------
Spigolo|  Lato   | Spigolo
```

Ora, chiunque sia "X", inizia per primo. La miglior mossa iniziale è prendere uno spigolo. Uno spigolo qualunque, non importa quale sia. Ce ne occuperemo con le combinazioni di gioco per la prima mossa di "X", che sono indicati destra.

Il punto di vista del giocatore "O" è mostrato in basso a destra.

```
SE “O” prende uno SPIGOLO ALLORA
     # Scenario 1
     “X” dovrebbe prendere uno degli spigoli rimasti. Non importa quale.
     SE “O” blocca la vittoria ALLORA
          “X” prende il restante spigolo.
          Termina per vincere.
    ALTRIMENTI
         Termnare per vincere.
ALTRO SE “O” prende un LATO ALLORA
     # Scenario 2
     “X” prende il CENTRO
     SE “O” blocca la vittoria ALLORA
          “X” prende lo spigolo che non è delimitato con nessun “O”
          Termina per vincere.
     ALTRIMENTI
          Termina per vincere.
ALTRIMENTI
     # “O” ha preso il centro – Scenario 3
     “X” prende lo spigolo diagonalmente opposto
alla mossa iniziale
     SE “O” prende uno spigolo
          “X” prende il restante spigolo
          Termina per vincere.
     ALTRIMENTI
          # Il gioco avrà un pareggio – Scenario 4
          Blocca la vittoria di “O”.
          Blocca ogni altra possibilità di vittoria
          Pareggio.
```

```
SE “X” non prende il centro ALLORA
     “O” prende il Centro
     SE “X” ha preso lo spigolo E il
lato ALLORA
          #Scenario 5
          “O” prende lo spigolo
diagonalmente opposto da quello di  “X”
          Blocca ogni possibile
vittoria per pareggiare.
     ALTRIMENTI
          # “X” ha preso due Lati –
Scenario 6
          “O” prende lo spigolo
confinante con entrambe le  “X”
          SE “X” blocca la vittoria
ALLORA
           “O” prende ogni lato.
         Bloccare e forzare il pareggio
          ALTRIMENTI
               Termina per vincere.
```

Alcune possibili giocate sono mostrate nella prossima pagina.

Come si può vedere, la logica è alquanto complessa, ma può essere facilmente ripartita in una serie di istruzioni IF (notare che ho utilizzato "Then", ma in Python non lo usiamo, utilizziamo invece ":"). Dovreste essere capaci di modificare il codice del mese scorso per adattarlo, o come minimo scriverne uno da zero per essere un semplice programma desktop Tris.

```
Scenario 1
 X | - | -    X | - | -    X | - | -    X | - | -    X | - | X    X | - | X    X | X | X
 - | - | -    - | - | -    - | - | -    O | - | -    O | - | -    O | O | -    O | O | -
 - | - | -    - | - | O    X | - | O    X | - | O    X | - | O    X | - | O    X | - | O

Scenario 2
 X | - | -    X | - | -    X | - | -    X | - | -    X | - | X    X | - | X    X | X | X
 - | - | -    O | - | -    O | X | -    O | X | -    O | X | -    O | X | -    O | X | -
 - | - | -    - | - | -    - | - | -    - | - | O    - | - | O    O | - | O    X | - | O

Scenario 3
 X | - | -    X | - | -    X | - | -    X | - | X    X | O | X    X | O | X    X | O | X
 - | - | -    - | O | -    - | O | -    - | O | -    - | O | -    O | - | -    O | - | X
 - | - | -    - | - | -    - | - | X    O | - | X    O | - | O    X | - | O    X | - | X

Scenario 4
 X | - | -    X | - | -    X | - | -    X | - | -    X | - | -    X | - | X    X | O | X
 - | - | -    - | O | -    - | O | O    X | O | O    X | O | O    X | O | O    X | O | O
 - | - | -    - | - | -    - | - | X    - | - | X    O | - | X    O | - | X    O | - | X

Scenario 5
 X | - | -    X | - | -    X | - | -    X | - | -    X | - | -    X | - | -    X | - | X
 - | - | -    - | O | -    - | O | X    - | O | X    X | O | X    X | O | X    X | O | X
 - | - | -    - | - | -    - | - | -    - | - | O    - | - | O    O | - | O    O | - | O

Scenario 6
 - | - | -    - | - | -    - | - | -    - | - | -    - | - | X    O | - | X    O | - | X
 X | - | -    X | O | -    X | O | -    X | O | -    X | O | -    X | O | -    X | O | -
 - | - | -    - | - | -    - | X | -    O | X | -    O | X | -    O | X | -    O | X | O
```

Greg Walters è proprietario della RainyDay Solution, LLC, una società di consulenza in Aurora, Colorado, e programma dal 1972. Gli piace cucinare, fare escursioni, la musica e trascorrere il tempo in famiglia.

Proposte e Autori Cercansi

Su LaunchPad abbiamo creato le pagine del progetto e della squadra Full Circle. L'idea è quella che i nonscrittori possono collegarsi alla pagina, fare clic su "Answers" in alto e lasciare **idee per articoli, ma vi prego siate specifici!** Non inserite solo "articolo sui server" ma indicate anche cosa il server dovrebbe fare!

I lettori che volessero scrivere un articolo ma sono a corto di idee, possono registrarsi alla pagina del gruppo Full Circle quindi auto-assegnarsi gli articoli proposti e iniziare a scrivere! Chiediamo che se non è possibile scrivere **l'articolo nel giro di alcune settimane (un mese circa) la richiesta venga riaperta** per permettere a qualcun'altro di adottarla.

- Pagina del progetto **per le idee**: https://launchpad.net/fullcircle
- Pagina per il gruppo **d'autori**: https://launchpad.net/~fullcircle

Programmare in Python - Parte 20

Bentornati. Questa volta torneremo a occuparci di GUI ma utilizzando la libreria pyGTK. Per il momento non useremo un designer di GUI, ma ricorreremo solo alla libreria.

Utilizzate Synaptic per installare python-gtk2, python-gtk2-tutorial e python-gtk2-doc.

Iniziamo subito con il primo programma che usa pyGTK, in alto a destra.

Per un po' ci concentremo su del codice semplice. La riga 3 contiene un nuovo comando. "pygtk.require('2.0')" significa che il programma verrà eseguito solo se il modulo pygtk installato è, come minimo, alla versione 2.0. Nella routine __init__ assegniamo una finestra alla variabile self.window (riga 8) e quindi la mostriamo (riga 9). Ricordate che la funzione __init__ è eseguita non appena la classe è istanziata (riga 13). Salvate il codice come "simple1.py".

Eseguitelo in un terminale.

```
# simple.py
import pygtk
pygtk.require('2.0')
import gtk

class Simple:
     def __init__(self):
          self.window = gtk.Window(gtk.WINDOW_TOPLEVEL)
          self.window.show()
     def main(self):
          gtk.main()
if __name__ == "__main__":
     simple = Simple()
     simple.main()
```

Vedrete comparire da qualche parte sul desktop una semplice finestra. Sul mio compare nell'angolo superiore sinistro. Per terminare il programma, dovete premere Ctrl+C nel terminale. Perché? Non abbiamo aggiunto il codice per distruggere e quindi terminare l'applicazione. Questo lo andiamo a fare ora. Aggiungete la seguente riga prima di self.window.show()...

```
self.window.connect("delete_event", self.delete_event)
```

Quindi, dopo la chiamata gtk.main(), aggiungete la seguente funzione...

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

Salvate la vostra applicazione come "simple2.py" e, ancora una volta, eseguitela dal terminale. Ora, quando cliccate su "X" nella barra del titolo, l'applicazione terminerà. Cosa è accaduto, quindi? La prima riga che abbiamo aggiunto (self.window.connect...) connette l'evento delete a una routine di servizio, in questo caso self.delete_event. Ritornando "False" si consente la rimozione della finestra dalla memoria di sistema.

Ora non so voi ma io preferisco che le applicazioni si aprano al centro dello schermo, non in una posizione a caso, o in un angolo dove potrebbe essere nascosta da qualcos'altro. Modifichiamo il codice di conseguenza. Tutto quello che dobbiamo fare è aggiungere la riga seguente prima di self.window.connect nella funzione __init__:

```
self.window.set_position(gtk.WIN_POS_CENTER)
```

Come potete ben immaginare, l'istruzione posiziona la finestra al centro dello schermo. Salvate l'applicazione come "simple3.py" ed eseguitela.

Ora è molto meglio, ma non è granché. Allora aggiungiamo un widget. Se ricordate i VECCHI articoli su Boa Constructor, i widget non sono altro che controlli predefiniti che possiamo aggiungere alla nostra finestra per fare delle cose. Uno dei controlli più semplici da aggiungere è un pulsante. Aggiungeremo il codice

seguente subito dopo self.window.connect nella routine __init__:

```
self.button =
gtk.Button("Close Me")
self.button.connect("clicked"
,self.btn1Clicked,None)

self.window.add(self.button)

self.button.show()
```

La prima riga definisce il pulsante e il relativo testo. La seconda è la connessione all'evento click. La terza aggiunge il pulsante alla finestra mentre la quarta riga lo mostra sulla superficie della finestra. Osservando self.button.connect noterete la presenza di tre argomenti. Il primo è l'evento a cui ci connettiamo, il secondo è la funzione eseguita quando l'evento si verifica, in questo caso "self.btn1Clicked" e il terzo è l'argomento (se esiste) che sarà passato alla funzione appena definita.

A seguire, dobbiamo creare la funzione self.btn1Clicked. Inseritela dopo self.delete_event:

```
def
btn1Clicked(self,widget,data=
None):
    print "Button 1 clicked"

    gtk.main_quit()
```

Come potete vedere, la routine non fa molto. Mostra nel terminale "Button 1 clicked" e quindi richiama la funzione gtk.main_quit(). Questa chiuderà la finestra e terminerà l'applicazione, come se aveste fatto clic su "X" sulla barra del titolo. Ancora, salvate il tutto come "simple4.py" ed eseguitelo in un terminale. Vedrete una finestra centrata con un pulsante con su scritto "Close me". Fate click e l'applicazione si chiuderà, come voluto. Notate, comunque, che la finestra è molto più piccola che in simple3.py. La potete ridimensionare ma questo farà ingrandire anche il pulsante. Perché? Semplicemente perché abbiamo inserito un pulsante nella finestra la quale si ridimensiona per adeguarsi al contenuto.

Abbiamo in una certa misura violato le regole di programmazione di GUI inserendo il pulsante direttamente nella finestra, senza usare un contenitore. Dovreste ricordare che negli articoli riguardanti Boa Contructor usammo box ridimensionatori (contenitori) per i nostri controlli. Dovremmo farlo anche se abbiamo un solo widget. Nel prossimo esempio aggiungeremo un HBox (un box orizzontale) per racchiudere il nostro pulsante e aggiungerne un altro. Se avessimo voluto un contenitore verticale avremmo usato VBox.

Per iniziare, usiamo simple4.py come base. Eliminate tutto tra le righe self.window.connect(...) e self.window.show(). Qui aggiungeremo le nuove righe. Il codice per HBox e il primo pulsante è...

```
self.box1 = gtk.HBox(False,0)

self.window.add(self.box1)

self.button =
gtk.Button("Button 1")

self.button.connect("clicked"
,self.btn1Clicked,None)

self.box1.pack_start(self.but
ton,True,True,0)

self.button.show()
```

Analizziamolo un po' alla volta. Abbiamo un HBox, chiamato self.box1. I parametri passati sono homogeneous (True o False) e un valore per lo spazio:

Proposte e Autori Cercansi

Su LaunchPad abbiamo creato le pagine del progetto e della squadra Full Circle. L'idea è quella che i nonscrittori possono collegarsi alla pagina, fare clic su "Answers" in alto e lasciare **idee per articoli, ma vi prego siate specifici!** Non inserite solo "articolo sui server" ma indicate anche cosa il server dovrebbe fare!

I lettori che volessero scrivere un articolo ma sono a corto di idee, possono registrarsi alla pagina del gruppo Full Circle quindi auto-assegnarsi gli articoli proposti e iniziare a scrivere! Chiediamo che se non è possibile scrivere **l'articolo nel giro di alcune settimane (un mese circa) la richiesta venga riaperta** per permettere a qualcun'altro di adottarla.
- Pagina del progetto **per le idee**: https://launchpad.net/fullcircle
- Pagina per il gruppo **d'autori**: https://launchpad.net/~fullcircle

```
HBox =
gtk.HBox(homogeneous=False,sp
acing=0)
```

Il parametro homogeneous controlla se ciascun widget nel box ha la stessa dimensione (larghezza nel caso di un HBox e altezza nel caso di un VBox). In questo caso passiamo false e un valore spazio di 0. A seguire aggiungiamo il box alla finestra. Quindi creiamo il pulsante come prima e colleghiamo l'evento click alla nostra funzione.

Ora arriviamo ad un nuovo comando. self.box1.pack_start è usato per aggiungere il pulsante al contenitore (HBox). Usiamo questo invece di self.window.add per i widget che vogliamo includere nel contenitore. Il comando (come sopra) è...

```
box.pack_start(widget,expand=
True, fill=True, padding=0)
```

Ha i seguenti parametri. Prima il widget, quindi expand (True or False), quindi fill (True or False) e un valore per il padding. Per i contenitori lo spacing rappresenta la quantità di spazio tra i widget mentre il padding si applica sul lato destro/sinistro del widget.

L'argomento expand ci permette di scegliere se il controllo dovrà riempire lo spazio extra nel box (True) o se il box dovrà restringersi per adattarsi al widget (False). L'argomento fill ha effetto solo se l'argomento expand è True. Per finire mostriamo il pulsante. Segue il codice per il secondo pulsante:

```
self.button2 =
gtk.Button("Button 2")

self.button2.connect("clicked
",self.btn2Clicked,None)

self.box1.pack_start(self.but
ton2,True,True,0)

self.button2.show()

self.box1.show()
```

Potete osservare come il codice sia molto simile al precedente. L'ultima riga mostra il box.

Ora dobbiamo aggiungere la funzione self.btn2Clicked. Dopo self.btn1Clicked inserite il seguente codice...

```
def
btn2Clicked(self,widget,data=
None):

    print "Button 2 clicked"
```

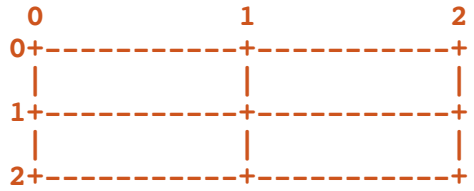
e in self.btn1Clicked commentate la riga:

```
gtk.main_quit()
```

Vogliamo che entrambi i pulsanti stampino il rispettivo "Button X clicked" senza chiudere la finestra.

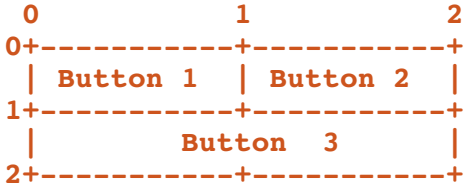
Salvate come "simple4a.py". Eseguitelo nel terminale. Vedrete una finestra centrata con due pulsanti (giusto ai bordi della finestra) etichettati "Button 1" e "Button 2". Fate clic su ciascuno e vedrete che risponderanno propriamente all'evento click come discusso. Ora, prima di chiudere la finestra, ridimensionatela (trascinate l'angolo in basso a destra) e noterete che i pulsanti si allargano e restringono seguendo il ridimensionamento della finestra. Per capire il parametro expand, cambiate il codice di entrambe le righe self.box1.pack_start da True a False. Riavviate il programma e osservate cosa accade. Questa volta, la finestra all'inizio sembra la stessa ma quando la ridimensionerete i pulsanti manterranno la dimensione iniziale con conseguente spazio vuoto a destra allargando la finestra. Proseguiamo ripristinando a True il valore di expand e impostiamo il parametro fill su False. Rieseguite e noterete che i pulsanti ancora manterranno la larghezza iniziale ma questa volta lo spazio bianco sarà distribuito a destra e a sinistra, ridimensionando la finestra. Ricordate che il parametro fill non fa nulla se expand è impostato su False.

Un altro modo per organizzare i widget è tramite l'uso di una tabella. Molte volte, se quello che abbiamo può essere disposto mediante una struttura a griglia allora la tabella è la scelta migliore (e più semplice). Immaginate la tabella come una griglia di un foglio di calcolo con righe e colonne contenenti widget. Ciascun widget può occupare una o più celle, come richiesto dalla vostra applicazione. Probabilmente il diagramma seguente aiuta a visualizzare le possibilità. Ecco una griglia 2x2:



Nella prima riga inseriremo due pulsanti, uno in ciascuna colonna. Nella seconda riga inseriremo un

pulsante che occuperà entrambe le colonne. Come questo...



Per impostare una tabella, creiamo un oggetto table e lo aggiungiamo alla finestra. La chiamata per creare la tabella è...

```
Table =
gtk.Table(rows=1,columns=1,ho
mogeneous=True)
```

Se la variabile homogeneous è uguale a True, la dimensione della tabella sarà quella del widget più grande della tabella stessa. Se impostato a False, la dimensione sarà determinata dal widget meno alto della stessa riga e da quello più largo nelle sua colonna. Quindi creiamo un widget (come il pulsante visto prima) e lo inseriamo nella tabella nella riga/colonna appropriata. La chiamata è come segue...

```
table.attach(widget,left
point,right point,top
point,bottom
point,xoptions=EXPAND|FILL,yo
ptions=EXPAND|FILL,xpadding=0
,ypadding=0)
```

Gli unici parametri richiesti sono i primi 5. Quindi per inserire un pulsante nella riga 0, colonna 0 potremmo usare la seguente istruzione...

```
table.attach(buttonx,0,1,0,1)
```

Se avessimo voluto inserirlo nella riga 0, colonna 1 (l'indice inizia da 0) come il pulsante 2 di prima, la chiamata sarebbe stata...

```
table.attach(buttonx,1,2,0,1)
```

Speriamo che questo sia chiaro, in un certo qual modo. Iniziamo con il codice vero e proprio e capirete meglio. Prima la parte in comune...

```
# table1.py
import pygtk
pygtk.require('2.0')
import gtk
class Table:
    def __init__(self):

        self.window =
gtk.Window(gtk.WINDOW_TOPLEVE
L)

self.window.set_position(gtk.
WIN_POS_CENTER)

self.window.set_title("Table
Test 1")

self.window.set_border_width(
20)

self.window.set_size_request(
250, 100)

self.window.connect("delete_e
vent", self.delete_event)
```

Ci sono alcune cose da chiarire prima di procedere. La riga 9 imposta il titolo della finestra a "Table Test 1". Usiamo la chiamata "set_border_width" per dare un bordo di 20px intorno l'intera finestra prima di posizionare ogni altro widget. Quindi forziamo la dimensione della finestra a 250x100 pixel usando la funzione set_size_request. A ancora senso? Ora creiamo la tabella e l'aggiungiamo alla finestra...

```
table = gtk.Table(2, 2, True)
# Create a 2x2 grid

self.window.add(table)
```

Proseguiamo creando il nostro primo pulsante, lo connettiamo con l'evento, lo inseriamo nella tabella e lo mostriamo...

```
button1 = gtk.Button("Button
1")

button1.connect("clicked",sel
f.callback,"button 1")

table.attach(button1,0,1,0,1)

button1.show()
```

Ora il pulsante numero 2...

```
button2 = gtk.Button("Button
2")

button2.connect("clicked",sel
f.callback,"button 2")

table.attach(button2,1,2,0,1)

button2.show()
```

Quasi tutto come per il primo pulsante, ma fate attenzione al cambiamento della chiamata table.attach. Notate anche che la funzione usata per gestire l'evento è chiamata "self.callback", ed è la stessa per entrambi i pulsanti. Per ora va bene. Capirete cosa stiamo facendo a breve.

Ora il terzo pulsante, Sarà il nostro "Chiudi":

```
button3 = gtk.Button("Quit")

button3.connect("clicked",sel
f.ExitApp,"button 3")

table.attach(button3,0,2,1,2)

button3.show()
```

Per finire, mostriamo la tabella e la finestra. Anche qui ricorriamo

alle funzioni main e delete usate precedentemente:

```
    table.show()

    self.window.show()

def main(self):

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

    gtk.main_quit()

    return False
```

Ora la parte divertente. Sia per il pulsante 1 che per il 2 abbiamo impostato quale funzione di gestione dell'evento "self.callback". Ecco il suo codice.

```
def
callback(self,widget,data=Non
e):

    print "%s was pressed"
%data
```

Quello che accade è che quando l'utente fa clic sul pulsante, viene generato l'evento click e viene inviato il dato fornito alla creazione della connessione. Per il pulsante 1 il dato inviato è "button 1" e per il pulsante 2 è "button 2". Tutto quello che facciamo è stampare "button x was pressed" nel terminale. Sono sicuro che ne capirete l'utilità quando combinato a una struttura IF | ELIF | ELSE.

Per finire, dobbiamo definire la funzione "ExitApp" per quando si fa clic sul pulsante "Quit"...

```
def ExitApp(self, widget,
event, data=None):

    print "Quit button was
pressed"

    gtk.main_quit()
```

Ed ora il codice main finale...

```
if __name__ == "__main__":

    table = Table()

    table.main()
```

Combiniamo tutto questo codice in una singola applicazione chiamata "table1.py". Eseguitela nel terminale.

Per riepilogare, quando si voglia usare pyGTK per creare un programma con GUI, i passi da seguire sono...

- Creare la finestra.
- Creare HBox, VBox o Table per contenere i widget.
- Inserire i widget (con il codice appropriato per box o tabelle).
- Mostrare i widget.
- Mostrare il box o la tabella.
- Mostrare la finestra.

Ora abbiamo molti strumenti e conoscenze per procedere ulteriormente. Tutto il codice è reperibile su Pastebin: http://fullcirclemagazine.pastebin.com/wnzRsXn9. Ci vediamo la prossima volta.

Greg Walters è proprietario della RainyDay Solution, LLC, una società di consulenza in Aurora, Colorado, e programma dal 1972. Gli piace cucinare, fare escursioni, la musica e trascorrere il tempo in famiglia.

Full Circle Podcast

Nell'episodio n. 15: Brainstorm, FUD e Media Player

* **Recensione:** numero 44 di FCM.
* **Notizie:** Brainstorm ideas, voti nel Software Centre, Fuduntu, Unity, Android e molto altro!
* **Giochi:** Humble Indie Bundle 2, Mass Effect, FreeCiv e Dropbox.

Dimensioni dei file:
OGG: 46.9Mb
mp3: 40.4Mb

Durata: 1hr 24min 34secondi
Pubblicato il: 13 gennaio 2011

http://fullcirclemagazine.org/

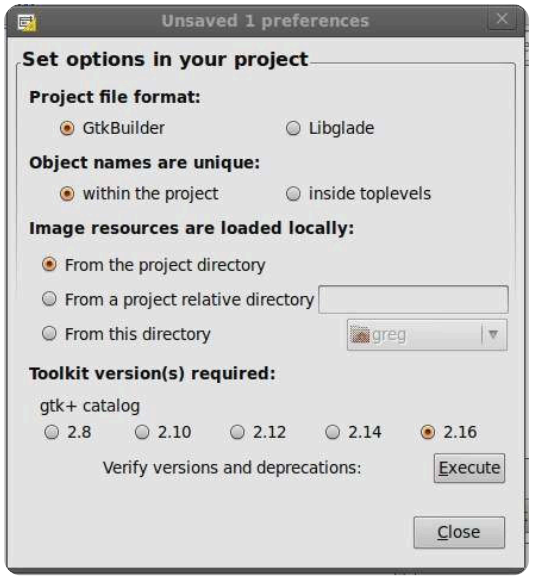
Programmare in Python - parte 21

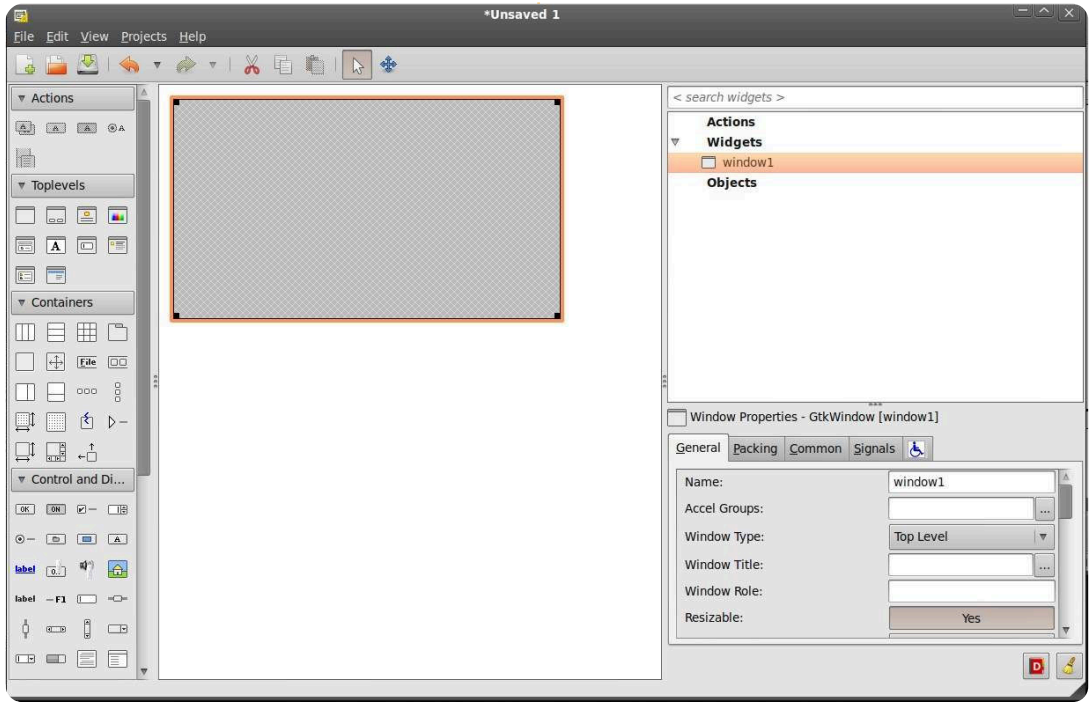
Se mi state seguendo da tempo, dovreste ricordare le lezioni 5 e 6. Allora parlammo di Boa Constructor e del suo utilizzo per realizzare applicazioni dotate di interfaccia grafica. Bene, questa volta impareremo a usare Glade. Differente, ma simile. Lo potete installare con Ubuntu Software Center: cercate glade ed installate Costruttore di interfacce utente GTK+ 2.

Giusto per la cronaca, l'applicazione di oggi sarà trattata in più lezioni. Il fine ultimo sarà realizzare un generatore di playlist per i nostri mp3 ed altri file multimediali. Questa parte del tutorial si focalizzerà sull'interfaccia. La prossima volta ci occuperemo del codice che terrà insieme le varie parti.

Iniziamo col disegnare la nostra applicazione. Una volta avviato Glade, si aprirà la finestra delle preferenze (sopra). Selezionate Libglade e "all'interno dei livelli principali", quindi fate clic su Chiudi. Ci sarà così mostrata la finestra principale.

Diamole un'occhiata (a destra). A sinistra c'è la tavolozza, in mezzo l'area del designer e a destra le aree





Proprietà e Ispettore.

Nella tavolozza, cercate il gruppo "Livelli principali" e fate clic sul primo strumento (posizionando il mouse comparirà la scritta "Finestra"). Otterremo una "lavagna" vuota con cui lavoreremo.

Notate che nell'Ispettore comparirà window1 sotto la sezione Widgets. Ora spostatevi nelle proprietà, cambiate il nome da window1 a MainWindow e impostate il Titolo della finestra a "Playlist Maker v1.0". Salvate il tutto come "PlaylistMaker.glade". Prima di proseguire, nella scheda Generale delle proprietà cercate il menù a discesa Posizione finestra e selezionate Centrato. Selezionate il checkbox Larghezza predefinita e impostate il valore a 650. Fate lo stesso con Altezza predefinita ma immettendo 350. Fate clic, quindi, sulla scheda Comuni e scorrete fino alla voce "Visibile". ASSICURATEVI CHE SIA SELEZIONATO "SÌ", altrimenti la vostra finestra non sarà visibile. Per finire, selezionate la scheda Segnali, scorrete alla sezione GtkObject e fate clic sulla freccia che punta a destra. In corrispondenza di destroy fate clic sul menù a discesa della colonna Gestore e selezionate "on_MainWindow_destroy". Questo creerà un evento che occorrerà quando l'utente farà clic sulla "X" della barra del titolo. Un avviso... Dopo aver impostato l'evento destroy, fate clic ovunque sopra o sotto per rendere effettivi i cambiamenti. Sembra si tratti di un bug di Glade. Ancora, salvate il progetto.

Proprio come allora, dobbiamo inserire i nostri widget all'interno di

vbox e hbox. Questa è la cosa più difficile da ricordare quando si programmano GUI. Andremo a inserire un contenitore verticale quindi nella tavolozza, nella sezione Contenitori, selezionate Casella verticale (seconda icona da sinistra nella riga superiore) e fate clic nella finestra vuota del designer. Comparirà una finestra con cui dovremo scegliere il numero di elementi desiderati. Il valore predefinito è tre, ma noi abbiamo bisogno di 5 elementi che serviranno, in ordine, per la barra strumenti, una lista ad albero, due sezioni per etichette, pulsanti e campi di testo, e, per finire, una barra di stato.

Possiamo cominciare ad aggiungere i widget. Per iniziare, aggiungiamo una barra degli strumenti dalla tavolozza. È (nella mia finestra) la quarta icona della seconda riga della sezione Contenitori. Fate clic sulla cella più in alto della nostra casella verticale. Noteremo che la cella si rimpicciolirà fino a quasi scomparire. Non preoccupatevi, la recupereremo a breve.

A seguire, dovremo aggiungere nella seconda cella una Finestra di scorrimento che conterrà la lista ad albero liberamente scorrevole. Quindi, cercate Finestra di scorrimento nella sezione Contenitori della tavolozza (seconda icona da sinistra nella quinta riga nella mia finestra) e fate clic sulla seconda cella della casella verticale. Proseguiamo inserendo due Caselle orizzontali, una per ciascuna delle due celle seguenti. Ciascuna dovrà contenere tre elementi. Per finire, aggiungete una Barra di stato nella cella in basso. La trovate verso la fine della sezione Controllo e visualizzazione della tavolozza. Ora il vostro designer dovrebbe assomigliare all'immagine in basso.

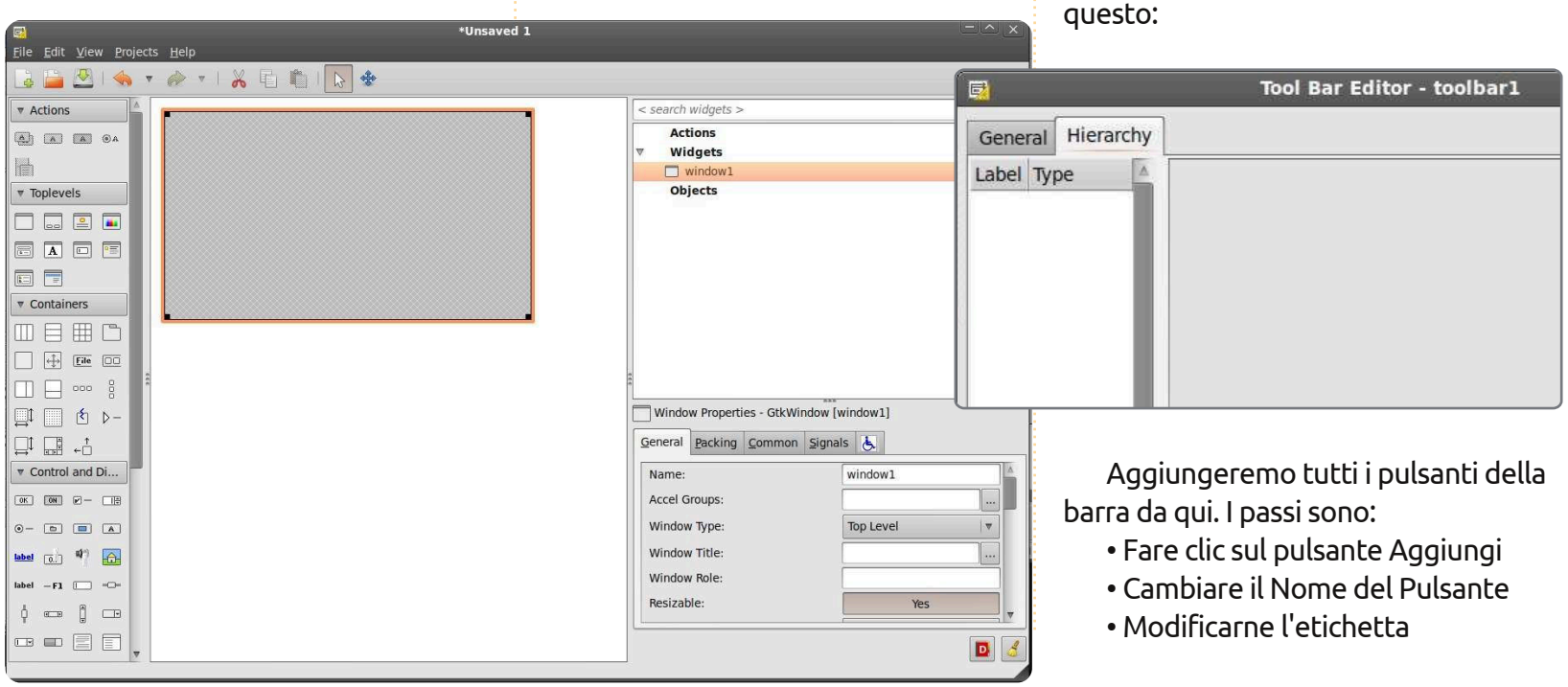
Ultimo, ma non meno importante: aggiungete una Vista albero dalla sezione Controllo e visualizzazione della tavolozza nel widget Finestra di scorrimento. Comparirà una finestra in cui scegliere il Modello TreeView da usare. Per ora semplicemente fate clic sul pulsante "OK". Lo imposteremo in seguito.

Ora dobbiamo concentrarci per un momento sulla Finestra di scorrimento. Fate clic su essa nell'ispettore. Scorrete la scheda Generale fino alla voce "Politica barra scorrimento orizzontale". Cambiare il valore in 'Sempre' e fate lo stesso per Politica barra scorrimento verticale. Salvate ancora.

Ok, concentriamoci ora sulla barra strumenti. Quest'area sarà quella più in alto, giusto sotto la barra del titolo. Conterrà vari pulsanti che ci permetteranno di eseguire la maggior parte delle operazioni. Useremo undici pulsanti e, da sinistra a destra, sono...

Aggiungi, Elimina, Pulisci lista, un Separatore, Sposta all'inizio, Sposta su, Sposta giù, Sposta alla fine, un altro Separatore, Informazioni e Esci

Nell'ispettore fate clic su "toolbar1" per selezionarlo. Nella parte superiore di Glade dovrebbe esserci qualcosa che somiglia a una matita. Fate clic su essa. Comparirà l'Editor della barra strumenti. Fate clic sulla scheda Gerarchia, vedrete qualcosa di simile a questo:



Aggiungeremo tutti i pulsanti della barra da qui. I passi sono:

- Fare clic sul pulsante Aggiungi
- Cambiare il Nome del Pulsante
- Modificarne l'etichetta

• Selezionare un'immagine

Ripeteremo la procedura per tutti gli undici widget. Quindi, fate clic su Aggiungi, nel campo Nome inserite "tbtnAdd". Scorrete in basso fino alla sezione Modifica etichetta e inserite "Aggiungi" in corrispondenza di Etichetta, quindi poco più in basso, sotto a Modifica immagine, in corrispondenza di ID dell'oggetto nello stock, usate il menù a discesa e selezionate "Aggiungi". Questo conclude il nostro pulsante Aggiungi. Lo abbiamo chiamato "tbtnAdd" così potremo riferirci ad esso in seguito nel codice. "tbtn" è l'abbreviazione di 'Toolbar Button'. In questo modo è più semplice da trovare nel codice, oltre a essere auto-documentante.

Dobbiamo ora aggiungere il resto dei widget alla barra strumenti. Aggiungiamo un altro pulsante per Elimina. Questo sarà chiamato (come potete ben immaginare) "tbtnDelete". Ancora, impostate l'etichetta e l'icona. A seguire, inserite un altro pulsante chiamato "tbtnClearAll" e usate l'icona Pulisci. Ora abbiamo bisogno del Separatore. Fate clic su Aggiungi, come nome inseriamo "Sep1" e nel menù a discesa Tipo scegliamo Separatore.

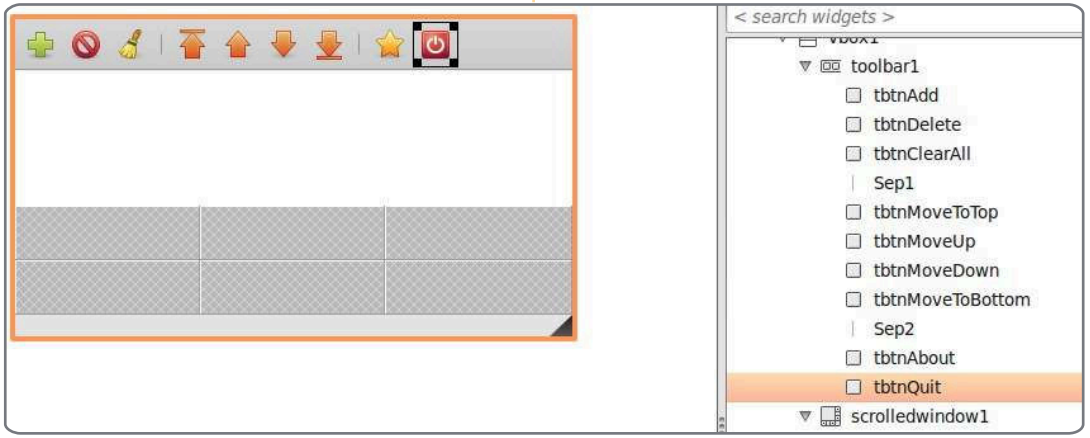
Aggiungiamo i widget rimanenti chiamandoli "tbtnMoveToTop", "tbtnMoveUp", "tbtnMoveDown", "tbtnMoveToBottom", "Sep2", "tbtnAbout" e "tbtnQuit". Sono sicuro che troverete le icone corrette. Una volta finito, potete chiudere la finestra delle Gerarchie [ndt: a dire il vero è la finestra Editor barra strumenti, Gerarchia è una sua scheda] e salvare il lavoro. Dovreste avere qualcosa che somiglia all'immagine in basso.

Ora dobbiamo impostare i gestori degli eventi per i pulsanti creati. Nell'ispettore, selezionate il widget tbtnAdd. Questa operazione dovrebbe evidenziare sia la voce nell'ispettore che il pulsante stesso. Tornate nell'area delle proprietà, selezionate la scheda Segnali ed espandete GtkToolButton per rivelare l'evento clicked. Nella colonna Gestore, come precedentemente, selezionate "on_tbtnAdd_clicked" quindi fate clic sopra o sotto per forzare la modifica. Fate lo stesso per tutti gli altri pulsanti creati, selezionando "on_tbtnDelete_clicked" e così via. Ricordate di fare clic all'esterno per forzare i cambiamenti e salvate il progetto. I separatori non hanno bisogno di eventi, quindi ignorateli.

A seguire, dobbiamo popolare le caselle orizzontali che conterranno ciascuna (da destra a sinistra) un'etichetta, un campo di testo ed un pulsante. Nella tavolozza, selezionate il widget Etichetta e inseritelo nella cella a sinistra. Poi inserite il widget Inserimento testo nella cella centrale e il pulsante in quella a destra. Ripetete la procedura per la seconda casella orizzontale.

Dobbiamo ora definire le proprietà dei controlli appena inseriti. Nell'ispettore, selezionate label1 sotto hbox1. Nell'area Proprietà, selezionate la scheda Generale, scorrete fino a "Modifica aspetto etichetta" e nella voce Etichetta inserite "Path to save file:". Quindi spostatevi nella scheda Posizionamento e impostate Espandere su "No". Dovreste ricordare il discorso sul posizionamento dalla lezione passata. Impostate il riempimento a 4 per inserire un po' di spazio a destra e sinistra dell'etichetta. Ora selezionate button1 e anche qui l'attributo Espandere della scheda Posizionamento va impostato su "No". Spostatevi sulla scheda Generale e come Nome inserite btnGetFolder. Notate che, visto che non si tratta di un pulsante della barra strumenti, abbiamo omesso la 't' iniziale. Scorrete alla voce Etichetta e inserite "Folder...". Quindi fate clic sulla scheda Segnali e impostate l'evento GtkButton/clicked su "on_btnGetFolder_clicked". Prima di impostare le proprietà degli elementi della successiva casella orizzontale, dobbiamo fare ancora una cosa. Selezionate hbox1 nell'ispettore e nella scheda Posizionamento impostate Espandere su "No". Questo per far sì che la casella occupi meno spazio. Per finire, impostate il nome del widget Inserimento testo su "txtPath".

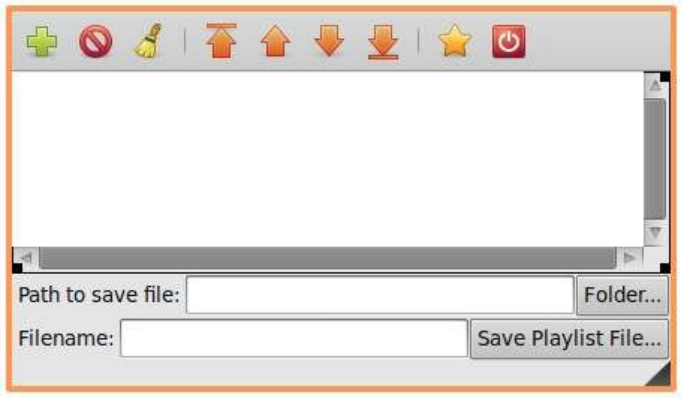
Ora fate lo stesso per la seconda casella orizzontale, impostando Espandere su "No", per l'etichetta



impostate il testo su "Filename:", Espandere su "No", e Riempimento a 4. Impostate il nome per il pulsante a "btnSavePlaylist", la sua etichetta a "Save Playlist File...", la proprietà Espandere su "No" e impostate l'evento clicked, quindi impostate il nome del widget Inserimento testo su "txtFilename". Ancora una volta, salvate tutto.

Ora la nostra finestra dovrebbe assomigliare all'immagine in basso a sinistra.

Tutto questo è magnifico, ma cosa fa davvero? Non possiamo eseguirlo, poiché non abbiamo nessun codice. Quello che abbiamo fatto è stato creare un file XML chiamato "playlistmaker.glade". Non fatevi ingannare dall'estensione. È proprio un file XML, lo potete verificare aprendolo con il vostro editor preferito (gedit nel mio caso) e osservarlo.




```
<widget class="GtkWindow" id="MainWindow">
    <property name="visible">True</property>
    <property name="title" translatable="yes">Playlist Maker v1.0</property>
    <property name="window_position">center</property>
    <property name="default_width">650</property>
    <property name="default_height">350</property>
    <signal name="destroy" handler="on_MainWindow_destroy"/>
```

Vedrete semplice testo che descrive la nostra finestra e ciascun widget con le relative proprietà. Per esempio, diamo un'occhiata al codice (sopra) per il widget principale, la finestra stessa.

Potete vedere che il nome del widget è "MainWindow", il suo titolo è "Playlist Maker v1.0", il gestore evento, e così via.

Osserviamo il codice (mostrato in basso) per un pulsante della barra strumenti.

Con un po' di fortuna dovrebbe incominciare ad avere senso. Ora dobbiamo scrivere un po' di codice per vedere all'opera il frutto del nostro duro lavoro. Avviate l'editor e iniziamo con questo...

Allora, abbiamo creato i nostri import molto similmente a quanto fatto il mese scorso. Notate che stiamo importando "sys" e "MP3" da mutagen.mp3. Abbiamo installato mutagen durante l'articolo n. 9 quindi se non è presente nel vostro sistema, fate riferimento a quel numero. Mutagen servirà la prossima volta, mentre sys è usato affinché il sistema possa terminare correttamente con l'ultima eccezione.

Quindi dobbiamo creare la classe che definirà la finestra. La trovate in alto a destra.

Simile a quella creata in precedenza. Osservate le ultime due righe. Stiamo definendo come nome del file glade (self.gladfile) quello creato con Glade. Notate anche che non abbiamo inserito il percorso, ma solo il nome del file. Se il file glade risiederà in un percorso differente da quello del codice, allora dovremo

```
<child>
   <widget class="GtkToolButton" id="tbtnAdd">
      <property name="visible">True</property>
      <property name="label" translatable="yes">Add</property>
      <property name="use_underline">True</property>
      <property name="stock_id">gtk-add</property>
      <signal name="clicked" handler="on_tbtnAdd_clicked"/>
   </widget>
   <packing>
       <property name="expand">False</property>
       <property name="homogeneous">True</property>
   </packing>
</child>
```

indicarlo. Comunque è più semplice tenere tutto insieme. A seguire, definiamo la finestra come self.wTree cui faremo riferimento ogniqualvolta avremo bisogno di operare su di essa. Stiamo anche dicendo che il file

```
#!/usr/bin/env python
import sys
from mutagen.mp3 import MP3
try:
    import pygtk
    pygtk.require("2.0")
except:
    pass
try:
    import gtk
    import gtk.glade
except:
    sys.exit(1)
```

utilizzato è di tipo XML e che la finestra userà l'elemento chiamato "MainWindow". Potete avere più finestre definite in un singolo file glade. Di più a riguardo, un'altra volta.

Ora dobbiamo occuparci degli eventi. Il mese scorso abbiamo usato button.connect o window.connect per far riferimento alle funzioni di gestione degli eventi. Questa volta faremo una cosa leggermente diversa. Useremo un dizionario. Un dizionario è come un array con la differenza che invece di usare un indice, useremo una chiave cui corrisponderà un valore. Chiave e Valore. Ecco il codice che probabilmente lo renderà più comprensibile. Per il momento vi fornirò solo due eventi (mostrati in basso)...

Allora, abbiamo due eventi: "on_MainWindow_destroy" e "on_tbtnQuit_clicked" sono le chiavi del nostro dizionario. Il valore è "gtk.main_quit" per entrambe. Ogniqualvolta dalla GUI viene richiamato l'evento, il sistema usa l'evento per trovare la chiave nel nostro dizionario e quindi sa quale funzione chiamare, dal corrispondente valore. Non ci resta che connettere il dizionario al gestore del segnall della finestra. Lo facciamo con la seguente riga di codice.

```
self.wTree.signal_autoconnect
(dict)
```

Abbiamo quasi finito. Abbiamo

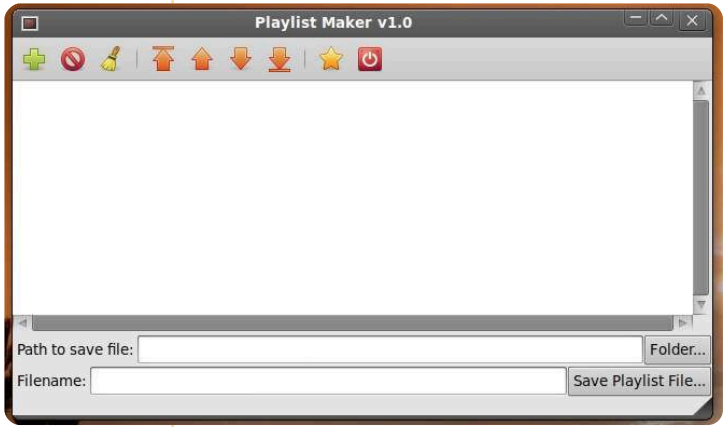
```
    #================================================
    #                  Create Event Handlers
    #================================================
    dict = {"on_MainWindow_destroy": gtk.main_quit,
            "on_tbtnQuit_clicked": gtk.main_quit}
```

```
class PlayListMaker:
    def __init__(self):
        #================================================
        #                        Window Creation
        #================================================
        self.gladefile = "playlistmaker.glade"
        self.wTree =
gtk.glade.XML(self.gladefile,"MainWindow")
```

ancora bisogno della routine principale:

```
if __name__ ==
"__main__":
    plm =
PlayListMaker()
    gtk.main()
```

Salvate il file come "playlistmaker.py". Ora potete eseguirlo (mostrato in alto a destra).



Al momento non fa molto oltre che aprirsi e chiudersi correttamente. Il resto sarà oggetto del prossimo articolo. Solo per stuzzicarvi l'appetito, discuteremo di come usare Viste albero, Finestre dialogo e aggiungeremo molto codice. Quindi sintonizzatevi alla prossima volta.

File glade:
http://fullcirclemagazine.pastebin.com/YM6U0Ee3

Sorgente python:
http://fullcirclemagazine.pastebin.com/wbfDmmBh

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