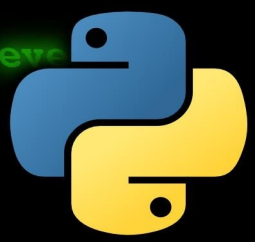
Full Circle

LA RIVISTA INDIPENDENTE PER LA COMUNITÀ LINUX UBUNTU

EDIZIONE SPECIALE SERIE PROGRAMMAZIONE

EDIZIONE SPECIALE
SERIE PROGRAMMAZIONE



python™

PROGRAMMARE IN PYTHON
VOLUME 8

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 44-48,** pubblicata nei numeri 73-78, consentendo all'impareggiabile professor Gregg Walters il nr.74 come tempo libero per buona condotta.

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

Buon divertimento!

Come contattarci

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

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

IRC:#fullcirclemagazine su chat.freenode.net

Gruppo editoriale

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

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

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

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

Programmare In Python: Parte 44

Questo mese faremo una piccola deviazione dal nostro programma TVRage per rispondere parzialmente alle domande di un nostro lettore. Mi era stato chiesto di parlare di QT Creator e di come usarlo per disegnare interfacce utente per i programmi Python.

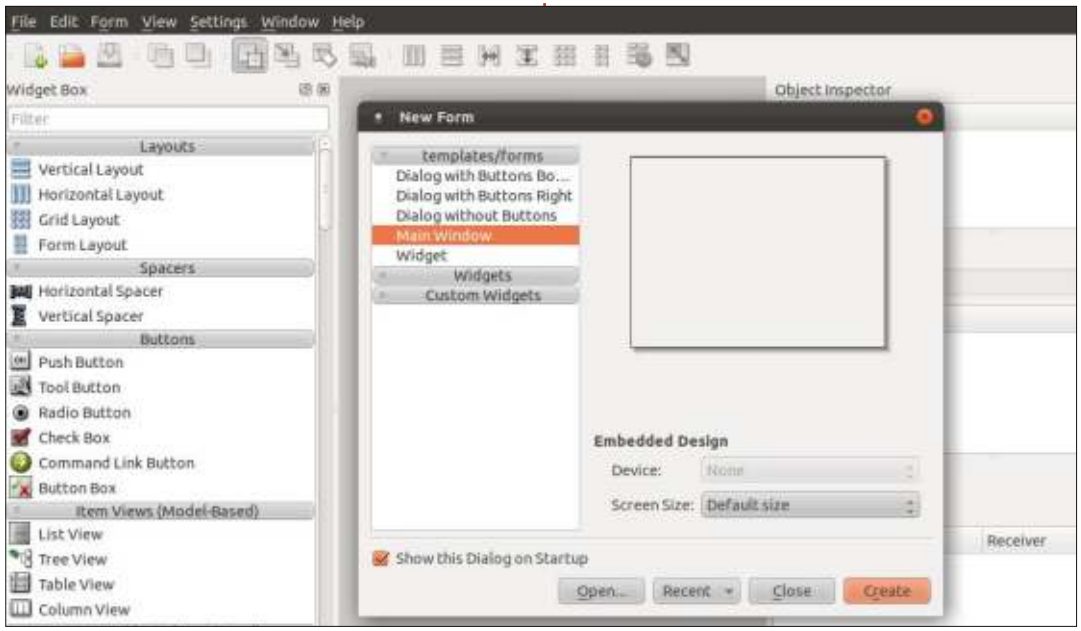
Sfortunatamente, da quello che posso dire, il supporto per QT Creator non è ancora pronto per Python. È in lavorazione, ma non è ancora del tutto 'pronto per il debutto'.

Così, nel tentativo di prepararci per questo futuro articolo, lavoreremo con QT4 Designer. Si dovranno installare (se non lo sono già) python-qt4, qt4-dev-tools, python-qt4-dev, pyqt4-dev-tools e libqt4-dev.

Una volta fatto, sotto ad Applicazioni | Sviluppo potrete trovare QT4 Designer. Proseguite e avviatelo. Si dovrebbe presentare con qualcosa di simile al seguente:

Assicuratevi che 'Main Window' sia selezionata e premete sul pulsante 'Create'. Ora avrete una scheda bianca nella quale trascinare e rilasciare i controlli.

La prima cosa che vogliamo fare è ridimensionare la finestra principale, a circa 500x300. Potete dire quanto è grande dando uno sguardo a Property Editor sotto le proprietà geometriche nella parte destra della finestra di Designer. Quindi, scorrere verso il basso la casella di riepilogo delle proprietà dell'editor fino a vedere 'windowTitle'. Cambiate il testo da 'MainWindow' a 'Python Test1'. Dovreste veder cambiare la finestra del titolo della propria finestra di Designer in 'Python Test1 - untitled*'.



Ora è un buon momento per salvare il progetto. Nominatelo 'pytest1.ui'. Successivamente, metteremo un pulsante nella nostra form. Sarà un pulsante di uscita per terminare il programma di prova. Sul lato sinistro della finestra di Designer ci sono tutti i controlli disponibili. Trovare la sezione 'Buttons' e trascinare nella scheda il controllo 'Push Button'. A differenza delle interfacce grafiche usate in passato, usando QT4 Designer non si devono creare griglie per contenere i controlli. Spostate il pulsante vicino al centro inferiore della scheda. Se si guardano le proprietà dell'editor sotto a geometrie, si vedrà qualcosa di simile a questo:

```
[(200,260), 97x27]
```

Quelle tra parentesi sono le posizioni X e Y dell'oggetto (il pulsante, in questo caso) sulla scheda, seguite dalla sua altezza e larghezza. Ho spostato il mio a 200,260.

Proprio sopra a esso c'è la proprietà di objectName, che, per impostazione predefinita, è impostata a 'pushButton'. Cambiarla in 'btnExit'. Scorrere ora la lista di Property Editor fino alla sezione 'QAbstractButton' e impostare la proprietà di 'text' in 'Exit'. È possibile vedere sulla propria scheda che il testo sul pulsante è cambiato.

Ora, aggiungere un altro pulsante e posizionarlo a 200,200. Cambiare la sua proprietà objectName in 'btnClickMe' e impostarne il testo a 'Premimi!'.

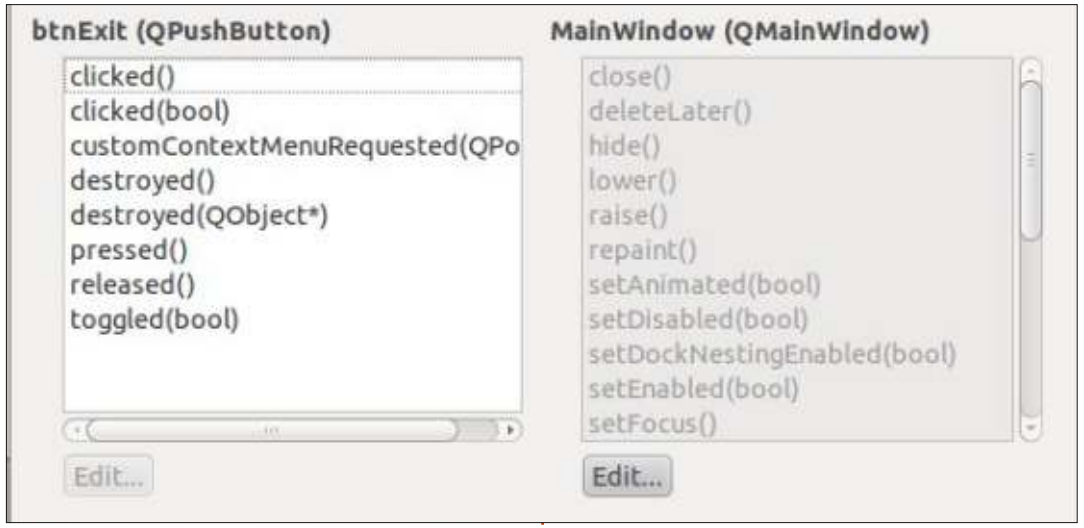
Aggiungete poi una etichetta. La troverete nella casella degli strumenti sulla sinistra sotto a 'DisplayWidgets'. Posizionatela vicino al centro della scheda (ho posizionato la mia a 210,130) e impostate la sua proprietà

objectName a 'lblDisplay'. Vogliamo renderla più grande di quanto sia per impostazione predefinita, quindi impostare le sue dimensioni all'incirca intorno a 221x20. Nell'editor delle proprietà, scorrete giù nella sezione 'Qlabel' e impostare l'allineamento orizzontale a 'AlignHCenter'. Cambiate il testo in spazio vuoto. Verrà impostato nel codice quando il pulsante btnClickMen sarà premuto. Ora salvate nuovamente il progetto.

Slot e Segnali

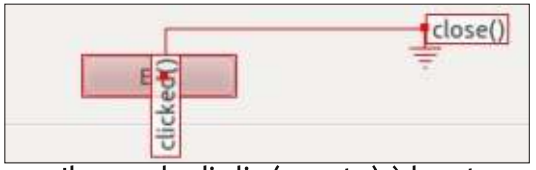
La prossima sezione potrebbe essere un po' più difficile da capire, specialmente se ci avete seguito per molto tempo e avete affrontato i precedenti disegnatori di interfacce grafiche. Negli altri disegnatori usavamo eventi che venivano attivati alla pressione di un oggetto, a esempio un pulsante. In QT4 Designer, gli eventi sono chiamati Segnali e la funzione che viene chiamata da questi Segnali è chiamata Slot. Quindi, per il pulsante Exit verrà usato il segnale Clic per chiamare lo slot che chiude la finestra principale. Siete totalmente confusi ora? È passato del tempo dalla mia prima volta con QT, ma inizia ad avere senso dopo un po'.

Fortunatamente, c'è un modo



molto facile per usare gli slot e i segnali predefiniti. Premendo il tasto F4 della tastiera si entrerà nella modalità Edit Signal e Slots Mode (per uscirne, premere F3). Ora, premete con il tasto sinistro sul pulsante Exit, mantenetelo e trascinatelo leggermente in alto a destra nella scheda principale, quindi rilasciatelo. Si vedrà apparire una finestra di dialogo che somiglia a quanto mostrato sopra.

Ciò fornirà un modo facile per collegare alla scheda il segnale su cui si è fatto clic. Selezionate la prima opzione a sinistra, che dovrebbe essere 'clicked()'. Questo abiliterà il lato destro della finestra. Selezionare dalla lista l'opzione 'close()' e fare quindi clic su 'OK'. Si vedrà qualcosa di somigliante a questo:



Il segnale di clic (evento) è legato alla funzione Close della finestra principale.

Il segnale di clic di btnClickMe verrà fatto nel codice.

Salvate il file ancora una volta. Uscite da QT4 Designer e aprite un terminale. Cambiate il percorso in quello in cui è stato salvato il file. Ora genereremo un file python usando lo strumento a linea di comando pyuic4. Questo leggerà il file .ui. Il comando sarà:

```
pyuic4 -x pytest1.ui -o
pytest1.py
```

Il parametro -x dice di includere il codice per avviare e mostrare l'interfaccia utente. Il parametro -o dice di creare il file piuttosto che visualizzarlo solo sullo schermo. Una cosa importante da notare qui. ASSICURATEVI di aver fatto tutto in QT4 Designer prima di creare il file python. Diversamente, sarà completamente riscritto e dovrete ricominciare da zero.

Una volta fatto, otterrete il file python. Apritelo con il vostro editor preferito.

Il file stesso è lungo soltanto 65 righe circa, commenti inclusi. Ci sono solo pochi controlli quindi non sarebbe potuto essere molto lungo. Non mostrerò una grande quantità di codice. Dovreste essere capaci di seguire la maggior parte del codice, oramai. Comunque verrà creato e aggiunto codice al fine di inserire la funzionalità per impostare il testo dell'etichetta.

La prima cosa da fare è copiare la riga di segnale e slot e modificarla. Da qualche parte intorno alla riga 47 ci dovrebbe essere il seguente codice:

```
QtCore.QObject.connect(self.btn
Exit,
QtCore.SIGNAL(_fromUtf8("clicke
d()")), MainWindow.close)
```

Copiatelo e, giusto sotto a esso, incollatelo. Poi modificatelo in:

```
QtCore.QObject.connect(self.btn
ClickMe,
QtCore.SIGNAL(_fromUtf8("clicke
d()")), self.SetLabelText)
```

Ciò creerà quindi la connessione segnale/slot alla funzione che imposterà il testo dell'etichetta. Sotto alla funzione retranslateUi, aggiungere il seguente codice:

```
def SetLabelText(self):

self.lblDisplay.setText(_fromUt
f8("That Tickles!!!"))
```

Ho ottenuto le informazioni dell'etichetta setText dalla linea di inizializzazione nella funzione setupUi.

Eseguite ora il codice. Ogni cosa dovrebbe funzionare come ci si aspetta.

Sebbene questo è un esempio MOLTO semplice, sono sicuro che siete abbastanza esperti per giocare con QT4 Designer e farvi un idea della potenza dello strumento.

Il prossimo mese faremo ritorno da questa nostra deviazione iniziando a lavorare sull'interfaccia utente per il programma TVRage.

Come sempre, il codice può essere reperito su pastbin presso http://pastebin.com/98fSasdb per il codice .ui e presso http://pastebin.com/yC30B885 per il codice python.

Arrivederci alla prossima volta.

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

La mia storia rapida

di Anthony Venable

Questa storia comincia agli inizi del 2010. Ero al verde al momento così stavo tentando di trovare un sistema operativo gratis. Mi serviva qualcosa che potevo avviare sul mio PC di casa. Avevo cercato su Internet, ma non avevo trovato nulla di utile per molto tempo. Ma un giorno ero da Barnes and Noble e vidi una rivista su Linux (benché avessi sentito di Linux prima, non avevo mai pensato che fosse qualcosa che sarei stato capace di usare). Quando avevo chiesto a persone che sapevo essere dei professionisti del computer, mi era stato detto che era per esperti e difficile da usare. Non avevo mai sentito cose positive su esso. Sono così stupito di non essermici imbattuto prima.

Quando ho letto la rivista, mi sono esposto a Ubuntu 9.10 - Karmic Koala. Suonava così bene, come se fosse esattamente quello che stavo cercando. Come risultato, ero molto emozionato e lo portai a casa e, con mia sorpresa, fu così facile installarlo sul mio PC che decisi di usarlo insieme a Window XP come sistema dual boot. Tutto ciò che feci fu inserire il CD nel lettore e le istruzioni erano passo-passo che avreste dovuto essere veramente lenti per non capire come impostare ogni cosa.

Da allora sono stato molto soddisfatto di Ubuntu in generale e sono stato in grado di controllare le ultime versioni quali la 10.04 (Maverick Meerkat) e la 10.10 Lucid Lynx. Sono impaziente per la 11.04 Natty Narwhal per come integra il multi-touch e altro ancora rispetto alla 10.04.

Questa esperienza dimostra ancora una volta come riesco a trovare le cose più fighe per caso.

Programmare in Python - Parte 45

Scritto da Greg Walters

Questa volta andremo a rivedere il nostro programma di database creato negli articoli precedenti (parti 41, 42 e 43). Nei prossimi articoli, quindi, utilizzeremo la libreria QT per creare l'interfaccia utente.

Per prima cosa diamo un'occhiata a come funziona l'applicazione già esistente. Ecco una visione d'insieme generale:

- Crea una connessione al database e, se necessario, crea il database stesso.
- Crea un cursore per il database.
- Crea la tabella se non esiste già.
- Assegna la cartella/e video a una variabile.
- Cerca all'interno della cartella/e i video.
- Ottiene il nome del file, il nome della serie, il numero della stagione e il numero dell'episodio.
- Controlla se l'episodio esiste già nel database.
- Se non è già presente, lo aggiunge al database con valore "-1" così come l'ID TvRage
- Successivamente cerca all'interno del database l'ID e lo status, se necessario, e aggiorna il database.

Ridisegneremo il database per far sì che includa un'altra tabella e per modificare la tabella dati già esistente. Per prima cosa creiamo la nostra nuova tabella chiamandola Series. Essa conterrà tutte le informazioni riguardo le serie TV che abbiamo nel nostro sistema. La nuova tabella includerà i seguenti campi:

- Pkid
- Series Name
- TvRage Series ID
- Number of seasons
- Start Date
- Ended Flag
- Country of origin
- Status of the series (ended, current, etc)
- Classification (scripted, "reality", etc)
- Summary of the series plot
- Genres
- Runtime in minutes
- Network
- Day of the week it airs
- Time of day it airs
- Path to the series

Possiamo usare la esistente routine MakeDataBase per creare la nostra nuova tabella. Prima del codice esistente, aggiungete il codice mostrato sopra a destra.

```
sql = 'CREATE TABLE IF NOT EXISTS Series (
        pkid INTEGER PRIMARY KEY AUTOINCREMENT,
        SeriesName TEXT,
        SeriesID TEXT,
        Seasons TEXT,
        StartDate TEXT,
        Ended TEXT,
        OriginCountry TEXT,
        Status TEXT,
        Classification TEXT,
        Summary TEXT,
        Genres TEXT,
        Runtime TEXT,
        Network TEXT,
        AirDay TEXT,
        AirTime TEXT,
        Path TEXT);'
    cursor.execute(sql)
```

L'istruzione SQL ("sql = …") dovrebbe essere tutta su una sola linea, ma viene qui spezzata per facilitare la compresione. Lasciamo la modifica della tabella già esistente per dopo.

Ora dobbiamo modificare la nostra routine WalkThePath per far sì che il nome della serie e il percorso vengano salvati all'interno della tabella della serie.

Sostituiamo la linea con il codice

```
sqlquery = 'SELECT count(pkid)
as rowcount from TvShows where
Filename = "%s";' % fl
```

con

```
sqlquery = 'SELECT count(pkid)
as rowcount from series where
seriesName = "%s";' % showname
```

Questo codice, (giusto per rinfrescare la vostra memoria) controllerà se abbiamo già memorizzato le serie nella tabella. Ora cerchiamo le due righe con il codice:

```
sql = 'INSERT INTO TvShows
(Series,RootPath,Filename,Seaso
n,Episode,tvrageid) VALUES
(?,?,?,?,?,?)'

cursor.execute(sql,(showname,
root,fl,season,episode,-1))
```

e sostituiamole con

```
sql = 'INSERT INTO Series
(SeriesName,Path,SeriesID)
VALUES (?,?,?)'

cursor.execute(sql,(showname,
root,-1))
```

Ciò ci permetterà di inserire il nome delle serie (showname), il percorso della serie e un "-1" come ID TvRage. Useremo il contrassegno "-1" per indicare che si devono reperire le informazioni sulla serie da TvRage.

Adesso rivediamo la routine WalkTheDatabase per estrarre quelle serie che sono prive di informazioni (SeriesID = -1) e aggiornare i campi.

Cambiare la stringa di query da:

```
sqlstring = "SELECT DISTINCT
series FROM TvShows WHERE
tvrageid = -1"
```

a

```
sqlstring = "SELECT
pkid,SeriesName FROM Series
WHERE SeriesID = -1"
```

Questo creerà un risultato che ci permetterà di inoltrare una richiesta per ogni serie. Adesso cerchiamo e sostituiamo le seguenti due linee

```
seriesname = x[0]
searchname =
string.capwords(x[0]," ")
```

con

```
pkid = x[0]

seriesname = x[1]

searchname =
string.capwords(x[1]," ")
```

Useremo il pkID per l'istruzione di aggiornamente. Successivamente dobbiamo modificare la chiamata alla routine UpdateDdatabase per fare in modo che includa pkID. Modifichiamo la riga

```
UpdateDatabase(seriesname,id)
```

in

```
UpdateDatabase(seriesname,id,pk
id)
```

e anche la linea

```
GetShowStatus(seriesname,id)
```

in

```
GetShowData(seriesname,id,pkid)
```

Che sarà una nuova routine creata sul momento.

Adesso, cambiamo la definizione

```
def GetShowData(seriesname,id,pkid):
    tr = TvRage()
    idcursor = connection.cursor()
    dict = tr.GetShowInfo(id)
```

```
 seasons = dict['Seasons']
 startdate = dict['StartDate']
 ended = dict['Ended']
 origincountry = dict['Country']
 status = dict['Status']
 classification = dict['Classification']
 summary = dict['Summary']
```

della routine UpdateDatabase da

```
def
UpdateDatabase(seriesname,id):
```

in

```
def
UpdateDatabase(seriesname,id,
pkid):
```

Ora è necessario modificare la stringa di query da

```
sqlstring = 'UPDATE tvshows SET
tvrageid = ' + id + ' WHERE
series = "' + seriesname + '"'
```

in

```
sqlstring = 'UPDATE Series SET
SeriesID = ' + id + ' WHERE
pkID = %d' % pkid
```

Ora è necessario creare la routine GetShowData (in alto).

Prenderemo le informazioni da TvRage e le inseriremo nella tabella Series.

Ricordiamo che stiamo creando una istanza della routine TvRage e un dizionario che contiene tutte le informazioni delle nostre serie.

Successivamente creeremo della variabili per memorizzare i dati per aggiornare la tabella (sopra)

Da non dimenticare che Genres è un sottoelemento e contiene una lista di uno o più generi. Fortunatamente quando abbiamo scritto il codice della routine TvRage, abbiamo creato una stringa che memorizza tutti i generi, indipendentemente dal loro numero,

per cui possiamo usare semplicemente la striga del genere:

```
genres = dict['Genres']

runtime = dict['Runtime']

network = dict['Network']

airday = dict['Airday']

airtime = dict['Airtime']
```

Infine creiamo la stringa di query per effettuare l'aggiornamento (in basso). Nuovamente dovrebbe trovarsi tutto in una singola linea di codice, ma l'ho diviso per renderlo più semplice.

La sezione {number}, ricordiamo, è simile alla opzione di formattazione "%s". Questo creerà la nostra stringa di query sostituendo {number} con i dati che desideriamo. Visto che abbiamo già definito questi campi come testo, vogliamo usare le doppie virgolette per racchiudere i dati che aggiungeremo.

Infine, li scriviamo nel database (sotto).

E questo è tutto per questa volta, la prossima volta continueremo con quanto accennato all'inizio dell'articolo. Fino ad allora, divertitevi.

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

```
    try:
        idcursor.execute(sqlstring)
    except:
        print "Error Adding Series Information"
```

```
sqlstring = 'Update Series SET Seasons = "{0}", StartDate = "{1}", Ended = "{2}",
OriginCountry = "{3}", Status = "{4}", Classification = "{5}",
Summary = "{6}", Genres = "{7}", Runtime = "{8}", Network = "{9}",
AirDay = "{10}",AirTime = "{11}"
WHERE pkID ={12}'.format(seasons,startdate,ended,
origincountry,status,classification,summary,
genres,runtime,network,airday,airtime,pkid)
```

Programmare In Python - Parte 46

Generalmente i miei articoli sono abbastanza lunghi. Comunque, a causa di alcuni problemi medici, l'articolo di questo mese sarà piuttosto breve (nel grande schema delle cose). Tuttavia, ci sforzeremo e continueremo la nostra serie sul programma di gestione dei media.

Una delle cose che il programma dovrà fare per noi è di farci sapere se ci siamo persi qualche episodio di qualsiasi serie inserita nel database. Ecco lo scenario. Abbiamo una serie, che chiameremo 'Lo spettacolo degli anni 80', lunga tre stagioni. Nella seconda stagione c'erano 15 episodi. Tuttavia, ne abbiamo solo 13 nella nostra libreria. Come scopriamo, programmaticamente, quali episodi sono mancanti?

Il modo più semplice è di usare le liste e i set. Abbiamo già usato le liste in alcuni articoli negli ultimi quattro anni, ma i Set sono un nuovo tipo di dati in questo how-to, quindi gli esamineremo per un instante. Secondo la 'documentazione ufficiale' di Python (docs.python.org), questa è la definizione di un set:

"*Un set è una raccolta non ordinata senza elementi duplicati. Un uso basilare include le membership testing e l'eliminazione di voci duplicate. Gli oggetti Set supportano inoltre le operazioni matematiche quali le unioni, le intersezioni, le differenze e le differenze simmetriche*".

Continuerò a usare l'esempio dalla pagina di documentazione per illustrare il processo.

```
>>> Basket =
['apple','orange','apple','pe
ar','orange','banana']

>>> fruit = set(basket)

>>> fruit

set(['orange','pear','apple',
'banana'])
```

Notate che nell'elenco originale che è stato assegnato alla variabile basket, apple e orange sono state inserite due volte, ma quando la assegniamo a un set, i duplicati vengono scartati. Ora, per usare il set che abbiamo appena creato, dobbiamo verificare se una voce di fruit (o qualcos'altro) è nel set. Possiamo usare l'operatore 'in'.

```
>>> 'orange' in fruit

True

>>> 'kiwi' in fruit

False

>>>
```

Ciò è abbastanza semplice e, fortunatamente, state iniziando a vedere dove va tutto ciò. Diciamo di avere una lista della spesa con un mucchio di frutta annotata e, mentre andiamo verso il negozio, vogliamo verificare ciò che ci manca, fondamentalmente le voci nella lista della spesa ma non quelle nel nostro carrello. Possiamo cominciare così.

```
>>> shoppinglist =
['orange','apple','pear','bana
na','kiwi','grapes']

>>> basket =
['apple','kiwi','banana']

>>> sl = set(shoppinglist)

>>> b = set(basket)

>>> sl-b

set(['orange', 'pear',
'grapes'])

>>>
```

Creiamo le nostre due liste, shoppinglist per quello che ci serve e basket per quello che abbiamo. Assegniamo ciascuna a un set e quindi usiamo l'operatore di set per le differenze (il segno meno) per ottenere le voci che sono nella lista della spesa ma non nel carrello.

Ora, usando la stessa logica, creeremo una routine (pagina successiva in basso a sinistra) che si occuperà dei nostri episodi mancanti. Chiameremo la nostra routine 'FindMissing' e gli passeremo due variabili. La prima è un intero che è impostato con il numero degli episodi della stagione e la seconda è un elenco che contiene i numeri degli episodi che abbiamo per quella stagione.

La routine, quando viene eseguita, stampa [5, 8, 15], che è corretto. Ora diamo uno sguardo al codice. La prima linea crea un set chiamato EpisodesNeeded usando un elenco di interi creati tramite la funzione range. Ci occorre dotare la funzione range del valore iniziale e di quello finale. Aggiungiamo 1 al valore maggiore di range per

ottenere il giusto elenco di valori da 1 a 15. Ricordiamoci che la funzione range in realtà inizia da 0, quindi quando gli diamo 16 (expected (15)+1) l'effettiva lista che range crea è da 0 a 15. Diciamo alla funzione range di partire da 1, quindi anche se l'intervallo è da 0 a 15, che è di 16 valori, vogliamo che 15 inizi a 1.

Successivamente creiamo un set dall'elenco che viene passato nella routine, che contiene i numeri degli episodi che abbiamo effettivamente.

Ora possiamo creare una lista usando l'operatore di set per le differenze sui due set. Lo facciamo in modo che possiamo ordinarlo con il metodo list.sort(). Potete certamente farvi restituire l'elenco, se volete, ma in questa iterazione della routine, vi basta stamparlo.

Bene, questo è tutto il tempo che il mio corpo può sostenere seduto davanti al computer, quindi vi lascio per questo mese, chiedendovi come potremmo utilizzare ciò nel nostro gestore dei media.

Vi auguro un buon mese e ci vediamo presto.

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

```
def FindMissing(expected,have):
    #==================================
    # 'expected' is the number of episodes we should have
    # 'have' is a list of episodes that we do have
    # returns a sorted list of missing episode numbers
    #==================================
    EpisodesNeeded = set(range(1,expected+1))
    EpisodesHave = set(have)
    StillNeed = list(EpisodesNeeded - EpisodesHave)
    StillNeed.sort()
    print StillNeed

FindMissing(15,[1,2,3,4,6,7,9,10,11,12,13,14])
```

Edizioni Speciali di Python:

http://fullcirclemagazine.org/issue-py01/

http://fullcirclemagazine.org/issue-py02/

http://fullcirclemagazine.org/python-special-edition-issue-three/

http://fullcirclemagazine.org/python-special-edition-volume-four/

http://fullcirclemagazine.org/python-special-edition-volume-five/

http://fullcirclemagazine.org/python-special-edition-volume-six/

Programmare In Python - Parte 47

Lo scorso mese abbiamo discusso dell'uso dei set per mostrarci i numeri degli episodi mancanti. È tempo di mettere in pratica il rozzo codice presentato.

Modificheremo una routine e ne scriveremo una nuova. Faremo prima la modifica. Nel file di lavoro usato negli ultimi mesi, troviamo la routine WalkThePath(filepath). La quarta e quinta riga dovrebbero essere:

```
efile =
open('errors.log',"w")

for root, dirs, files in
os.walk(filepath,topdown=True):
```

Inseriremo il seguente codice fra queste due righe:

```
lastroot = ''

elist = []

currentshow = ''

currentseason = ''
```

Oramai dovreste riconoscere che ciò che abbiamo fatto qui è inizializzare le variabili. Ci sono tre variabili stringa e una lista. Useremo la lista per tenerci i numeri degli episodi (da qui il nome elist).

```
for root, dirs, files in os.walk(filepath,topdown=True):
    for file in [f for f in files if f.endswith (('.avi','mkv','mp4','m4v'))]:
```

Diamo una rapida occhiata e rinfreschiamoci la memoria (sopra) riguardo a ciò che stiamo facendo con la routine esistente, prima di modificarla ulteriormente.

Le prime due righe impostano le cose per la routine walk-the-path nella quale iniziamo in una data cartella del file system e visitiamo ricorsivamente ogni sotto-cartella e controlliamo i file che hanno estensione .avi, .mkv, .mp4 o .m4v. Se ce ne è qualcuno, allora lo reiteriamo nell'elenco di quei nomi di file.

Nella linea in alto a destra, chiamiamo la routine GetSeasonEpisode per estrarre il nome della serie, la stagione e il numero di episodio dal nome del file. Se tutto viene analizzato correttamente, la variabile isok viene impostata a true e le informazioni che stiamo cercando

```
# Combine path and filename to create a single variable.
fn = join(root,file)
OriginalFilename,ext = os.path.splitext(file)
fl = file
isok,data = GetSeasonEpisode(fl)
```

vengono messe nell'elenco e poi restituite.

Qui (sotto) assegniamo semplicemente i dati restituiti da GetSeasonEpisode e li inseriamo in variabili separate con cui possiamo giocare. Ora che sappiamo dove siamo, parliamo di dove andremo.

Vogliamo ottenere il numero di episodio di ciascun file e inserirlo nell'elenco elist. Una volta fatto con tutti i file dentro alla cartella corrente, possiamo allora ipotizzare che stiamo praticamente mantenendo il passo con i file e che l'episodio con il numero più alto è l'ultimo disponibile. Come abbiamo detto lo scorso mese, possiamo quindi creare un set numerato da 1 all'ultimo episodio e convertire l'elenco in un set ed estrarre la differenza. Sebbene ciò sia in teoria magnifico, c'è un piccolo intoppo nel nostro essere sulla giusta strada' quando si tratta di pratica reale. In realtà non otteniamo un'indicazione chiara e pulita quando abbiamo finito con una particolare cartella. Quello che abbiamo, però, è la consapevolezza che quando abbiamo finito con ciascun file, il codice subito dopo "for file in [..." viene eseguito. Se sappiamo il nome dell'ultima cartella visitata e

```
if isok:
    showname = data[0]
    season = data[1]
    episode = data[2]
    print("Season {0} Episode {1}".format(season,episode))
```

di quella attuale, possiamo comparare le due e, se sono diverse, abbiamo finito con una cartella e la nostra lista di episodi dovrebbe essere completa. A questo serve la variabile 'lastroot'.

Metteremo la maggior parte del nostro nuovo codice proprio dopo la riga 'for file in ['. Sono solo sette righe, eccole (le righe nere sono quelle esistenti, per vostra convenienza).

```
    for file in [f for f in files if f.endswith (('.avi','mkv','mp4','m4v'))]:
        # Combine path and filename to create a single variable.
        if lastroot != root:
            lastroot = root
            if len(elist) > 0:
                Missing(elist,max(elist),currentseason,currentshow)
            elist = []
            currentshow = ''
            currentseason = ''
        fn = join(root,file)
```

Riga per riga del nuovo codice, ecco la logica:

Innanzitutto, controlliamo per vedere se la variabile lastroot ha lo stesso valore di root (il nome dell'attuale cartella). Se così, siamo nella stessa cartella, quindi non dobbiamo eseguire nessun codice. Altrimenti, assegniamo il nome della cartella attuale alla variabile lastroot. Successivamente, controlliamo per vedere se l'elenco degli episodi (elist) ha qualche immissione (len(elist) > 0). Questo per assicurarsi che non eravamo in una cartella vuota. Se abbiamo voci nell'elenco, allora chiamiamo la routine Missing. Passiamo l'elenco episodi, il numero più alto dell'episodio, il numero della stagione corrente e il nome della stagione, quindi possiamo in seguito stamparli. Le ultime tre righe puliscono l'elenco, il nome attuale dello spettacolo e la stagione attuale, e andiamo avanti come abbiamo fatto prima.

Poi dobbiamo cambiare due righe e aggiungerne una nel codice if isok:, pochi righe sotto. Di nuovo, a destra, le righe nere sono il codice esistente:

```
        isok,data = GetSeasonEpisode(f1)
        if isok:
            currentshow = showname = data[0]
            currentseason = season = data[1]
            episode = data[2]
            elist.append(int(episode))
        else:
```

Qui siamo appena tornati indietro dalla routine GetSeasonEpisode. Se avevamo un nome di file analizzabile, vogliamo prendere il nome dello spettacolo e il numero di stagione e aggiungere l'episodio attuale all'elenco. Attenzione, convertiremo il numero dell'episodio in un numero intero prima di aggiungerlo all'elenco.

Abbiamo fatto con questa porzione di codice. Ora, quello che dobbiamo fare è di aggiungere la routine Missing. Proprio dopo la routine WalkThePath, aggiungeremo il codice in fondo pagina.

Anche in questo caso, è un insieme di codice molto semplice e lo abbiamo praticamente esaminato il mese scorso, ma faremo un ripasso nel caso ve lo foste perso.

Definiamo la funzione e impostiamo quattro parametri. Passeremo l'elenco degli episodi (eplist), il numero degli episodi che ci aspettiamo (shouldhave) che è il numero più alto di episodio nell'elenco episodi, il numero della stagione (season) e il nome dello spettacolo (showname).

Successivamente, creiamo un set che contiene un elenco di numeri usando la funzione nativa range, iniziando con 1 e andando nel valore

```
#--------------------------------
def Missing(eplist,shouldhave,season,showname):
    temp = set(range(1,shouldhave+1))
    ret = list(temp-set(eplist))
    if len(ret) > 0:
        print('Missing Episodes for {0} Season {1} - {2}'.format(showname,season,ret))
```

shouldhave +1. Chiamiamo quindi la funzione difference, su questo set e su un set convertito dall'elenco episodi (temp-set(eplist)), e lo riconvertiamo in un elenco. Controlliamo poi per vedere se c'è qualcosa nell'elenco, quindi non stampiamo una riga con un elenco vuoto e, se c'è qualcosa, la stampiamo.

Questo è quanto. L'unica crepa in questa logica è che facendo le cose in questa maniera non sappiamo se ci sono nuovi episodi che non abbiamo.

Ho messo le due routine su pastebin per voi, se volete solo fare una rapida sostituzione nel vostro codice funzionante. Le potete trovare presso http://pastebin.com/XHTRv2dQ.

Passate un buon mese, ci vediamo presto.

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

ubuntu uk podcast

Il Podcast Ubuntu copre tutte le ultime notizie e novità che si presentano agli utenti di Ubuntu Linux e ai fan del software libero in generale. La rassegna è rivolta tanto all'utente più fresco quanto al programmatore più esperto. Le nostre discussioni riguardano lo sviluppo di Ubuntu ma non sono eccessivamente tecniche. Siamo abbastanza fortunati da avere qualche gradito ospite nello show a passarci novità di prima mano sugli ultimi eccitanti sviluppi a cui stanno lavorando, in modo comprensibile a tutti! Parliamo inoltre della comunità Ubuntu e di cosa le interessa.

Lo show è offerto dai membri della comunità Ubuntu Linux del Regno Unito. Ed essendo coperto dal Codice di condotta di Ubuntu è adatto a tutti.

Lo show è trasmesso live ogni due settimane il martedì sera (ora inglese) ed è disponibile per il download il giorno seguente.

podcast.ubuntu-uk.org

Python Edizioni Speciali:

http://fullcirclemagazine.org/issue-py01/

http://fullcirclemagazine.org/issue-py02/

http://fullcirclemagazine.org/python-special-edition-issue-three/

http://fullcirclemagazine.org/python-special-edition-volume-four/

http://fullcirclemagazine.org/python-special-edition-volume-five/

http://fullcirclemagazine.org/python-special-edition-volume-six/

Programmare in Python - Parte 48

Ben tornati. È difficile immaginare che sono passati 4 anni da quando ho iniziato questa serie. Stavo pensando di mettere da parte per un po' il progetto di gestione dei media e di ritornare ad alcune basi della programmazione in Python.

Questo mese, rivisiterò il comando print. È uno delle funzioni più usate (almeno nella mia programmazione) che sembra non ottenere la descrizione che merita. Ci sono molte cose che si possono fare con esso oltre ai soliti '%s %d'.

Poiché la sintassi di print è differente tra Python 2.x e 3.x, gli daremo un'occhiata separatamente. Ricordatevi, comunque, che potete usare la sintassi del 3.x in Python 2.7. La maggior parte di ciò che presento questo mese sarà fatto dalla shell interattiva. Potete seguire passo passo. Il codice sarà come il seguente:

```
>>> a = "Hello Python"

>>> print("String a is %s"
%a)
```

e l'output sarà in grassetto, come questo:

```
String a is Hello Python
```

Python 2.x

Certamente vi ricordate della semplice sintassi della funzione print nella 2.x che utilizza la sostituzione di variabile %s o %d per le stringhe semplici o i decimali. Ma sono disponibili molte altre opzioni. Per esempio, se dovete formattare un numero con gli zeri iniziali, lo dovete fare in questo modo:

```
>>> print("Your value is
%03d" % 4)
Your value is 004
```

In questo caso, usiamo il comando di formattazione '%03d' per dire "mostra il numero con una estensione di tre caratteri e, se necessario, con gli zero davanti".

```
>>> pi = 3.14159
>>> print('PI = %5.3f.' % pi)
PI = 3.142.
```

Qui ho usato l'opzione per formattare il tipo float. Con '%5.3f' gli si dice di produrre un output con un totale di cinque caratteri e tre posti decimali. Notate che il punto del decimale si prende uno dei posti del totale dei caratteri.

Un'altra cosa che potreste non capire è che potete usare le chiavi di un dizionario come parte del comando di formattazione.

```
>>> info =
{"FName":"Fred","LName":"Fark
el","City":"Denver"}

>>> print('Greetings
%(FName)s %(LName)s of
%(City)s!' % info)

Greetings Fred Farkel of
Denver!
```

La tabella seguente mostra le varie chiavi di sostituzione possibili e il loro significato.

Conversion	Meaning
'd'	Signed integer decimal
'i'	Signed integer decimal
'u'	Obsolete - identical to 'd'
'o'	Signed octal value
'x'	Signed hexadecimal - lowercase
'X'	Signed hexadecimal - uppercase
'f'	Floating point decimal
'e'	Floating point exponential - lowercase
'E'	Floating point exponential - uppercase
'g'	Floating point format - uses lowercase exponential format if exponent is less than -4 or not less than precision, decimal format otherwise
'G'	Floating point format - uses uppercase exponential format if exponent is less than -4 or not less than precision, decimal format otherwise
'c'	Single character
'r'	String (converts valid Python object using repr())
's'	String (converts valid Python object using str())
'%'	No argument is converted, results in a '%' character

PYTHON 3.X

Con Python 3.x, abbiamo molte altre opzioni quando si tratta della funzione print (ricordatevi che potete usarle con Python 2.7).

Per rinfrescarvi la memoria, ecco un semplice esempio della funzione print con la 3.x.

```
>>> print('{0}
{1}'.format("Hello","Python")
)
Hello Python

>>> print("Python is {0}
cool!".format("WAY"))
Python is WAY cool!
```

I campi di sostituzione sono racchiusi nelle parentesi graffe "{" "}". Ogni cosa al di fuori di queste viene considerata un'espressione letterale e sarà stampata come tale. Nel primo esempio, abbiamo numerato i campi di sostituzione 0 e 1. Questo dice a Python di prendere il primo valore (0) e metterlo nel campo {0} e così via. Comunque, non dovete affatto usare tutti i numeri. Usando questa opzione si causa il posizionamento del primo valore nel primo insieme di parentesi e così via.

```
>>> print("This version of {}
is
{}".format("Python","3.3.2"))
This version of Python is
3.2.2
```

Come dicono nelle pubblicità in TV, "MA ASPETTATE... C'È DELL'ALTRO". Se vogliamo fare alcune formattazioni in linea, abbiamo le seguenti opzioni.

```
:<x Left align with a width
of x
:>x Right align with a width
of x
:^x Center align with a width
of x
```

Ecco un esempio:

```
>>>
print("|{:<20}|".format("Left
"))
|Left                |
>>>
print("|{:>20}|".format("Righ
"))
|               Right|
>>>
print("|{:^20}|".format("Cent
er"))
|       Center       |
```

Potete anche specificare un carattere di riempimento insieme al giustificato/ampiezza.

```
>>>
print("{:*>10}".format(321.40
))
*****321.4
```

Se dovete formattare un output di data/orario, potete fare qualcosa di simile a questo:

```
>>> d =
datetime.datetime(2013,10,9,1
0,45,1)

>>>
print("{:%m/%d/%y}".format(d)
)
10/09/13

>>>
print("{:%H:%M:%S}".format(d)
)
10:45:01
```

Stampare il separatore di migliaia usando la virgola (o qualsiasi altro carattere) è facile.

```
>>> print("This is a big
number
{:,}".format(7219219281))
This is a big number
7,219,219,281
```

Bene, questo dovrebbe darvi abbastanza cibo per la mente per questo mese. Ci vedremo all'inizio del quinto anno.

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

Il Podcast Ubuntu copre tutte le ultime notizie e novità che si presentano agli utenti di Ubuntu Linux e ai fan del software libero in generale. La rassegna è rivolta tanto all'utente più fresco quanto al programmatore più esperto. Le nostre discussioni riguardano lo sviluppo di Ubuntu ma non sono eccessivamente tecniche. Siamo abbastanza fortunati da avere qualche gradito ospite nello show a passarci novità di prima mano sugli ultimi eccitanti sviluppi a cui stanno lavorando, in modo comprensibile a tutti! Parliamo inoltre della comunità Ubuntu e di cosa le interessa.

Lo show è offerto dai membri della comunità Ubuntu Linux del Regno Unito. Ed essendo coperta dal Codice di condotta di Ubuntu è adatta a tutti.

Lo show è trasmesso live ogni due settimane il martedì sera (ora inglese) ed è disponibile per il download il giorno seguente.

podcast.ubuntu-uk.org