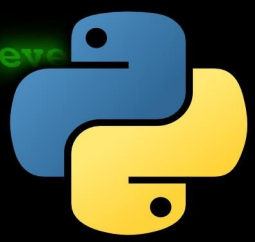
Full Circle

LA RIVISTA INDIPENDENTE PER LA COMUNITÀ LINUX UBUNTU

EDIZIONE SPECIALE SERIE PROGRAMMAZIONE

EDIZIONE SPECIALE
SERIE PROGRAMMAZIONE



python™

PROGRAMMARE IN PYTHON
VOLUME 7

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 39-43,** pubblicata nei numeri 68-72 dall'impareggiabile professor Gregg Walters.

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 39

Scritto da Greg Walters

Molti mesi fa abbiamo lavorato con le chiamate API per Weather Underground. Veramente, era nella parte 11 che era nel numero 37. Dunque avremo ancora a che fare con le API, questa volta di un sito denominato TVRage (http://tvrage.com). Se non siete familiari con questo sito, ha a che fare con gli show televisivi. Finora, ogni show televisivo che mi passava per la mente era anche presente nel loro sistema. In questa serie di articoli rivedremo XML, API e ElementTree (ndt Elementi ad Albero) per creare una libreria wrapper che ci permetterà di creare una piccola libreria che semplifica il nostro recupero delle informazioni TV sui nostri programmi preferiti.

Ora ho menzionato una libreria wrapper. Che cosa è? In termini semplici quando voi create o usate una libreria wrapper voi state usando un insieme di codice che ingloba la complessità delle API del sito web in una libreria facile da usare. Prima cosa, questo è un servizio gratuito. Comunque il sito richiede donazioni per l'uso delle loro API. Se valutate che questo è un servizio che vale, per favore considerate la donazione di 10 dollari americani o più. Seconda cosa, dovreste registrarvi al loro sito web e ottenere la vostra chiave per le API. È gratuito, quindi non c'è davvero nessuna ragione per non farlo, specialmente se voi state per usare le informazioni qui fornite. Inoltre, avrete accesso ad alcuni altri campi di informazioni come le serie e i sommari degli episodi che non sono inclusi nella versione non registrata. Terzo, essi fanno un lavoro pesante per aggiornare le API. Questo significa che quando voi starete leggendo questo articolo, le loro API potrebbero essere cambiate. Utilizzeremo i feed pubblici, che sono gratuiti per tutti, del Dicembre 2012. Il sito delle API è http://services.tvrage.com/info.php?page=main e mostra alcuni esempi di informazioni che sono disponibili.

Ora iniziamo a guardare alle API e come possiamo usarle.

Usando la loro API, possiamo ottenere delle informazioni molto specifiche sullo spettacolo stesso e/o possiamo ottenere informazioni sul singolo episodio. Ci sono fondamentalmente tre passi per cercare informazioni sugli spettacoli televisivi. Qui sono i passi:

- Cerca nel loro database usando il nome dello spettacolo per ottenere lo specifico Show ID che deve essere usato per ottenere ulteriori dati. Pensate al valore di showid come ad una chiave che punta direttamente in un insieme di record di un database.
- Una volta che avete lo Show ID, ottenete le informazioni di livello dello spettacolo.
- Infine raccogliete le informazioni riguardo allo specifico episodio. Queste provengono da una lista di tutti gli episodi in cui lo spettacolo è stato programmato.

Ci sono tre chiamate web di base che faremo per ottenere queste informazioni. La prima è la chiamata per la ricerca, la seconda per le informazioni sullo spettacolo e infine la chiamata per la lista degli episodi.

Ecco qui le chiamate di base che useremo…

- ricerca per ShowID basata sul nome dello spettacolo - `http://services.tvrage.com/feeds/search.php?show={SomeShow}`

- Estrazione delle informazioni a livello di spettacolo in base allo ShowID - `http://services.tvrage.com/feeds/showinfo.php?sid={SomeShowID}`

- Estrazione della lista degli episodi per lo ShowID (sid) - `http://services.tvrage.com/feeds/episode_list.php?sid={SomeShowID}`

Quello che viene restituito è un flusso di dati in formato XML. Prendiamoci un po' di tempo per rivedere come l'XML appare. La prima linea deve sempre essere simile a quella mostrata sotto per essere

```
<?xml version="1.0" encoding="UTF-8" ?>
<ROOT TAG>
   <PARENT TAG>
      <CHILD TAG 1>DATA</CLOSING CHILD TAG 1>
      <CHILD TAG 2>DATA</CLOSING CHILD TAG 2>
      <CHILD TAG 3>DATA</CLOSING CHILD TAG 3>
   </CLOSING PARENT TAG>
</CLOSING ROOT TAG>
```

considerata come un flusso di dati XML corretto (sotto).

Ciascun pezzo di dato è incluso all'interno di un tag (ndt contrassegno) di definizione e un tag di fine. Qualche volta avrete un tag figlio che è all'interno di un tag genitore come questo ...

```
<CHILD PARENT TAG>

<CHILD TAG 1>DATA</CLOSING
CHILD TAG 1>

</CLOSING CHILD PARENT TAG>
```

Potete anche vedere che un tag che ha un attributo associato:

```
<TAG INFORMATION = VALUE>

<CHILD TAG>DATA</CLOSING CHILD
TAG>

</CLOSING TAG>
```

Qualche volta , potreste vedere un tag con nessun dato associato ad esso. In questo caso sarebbe come questo ...

```
<prodnum/>
```

Qualche volta se non c'è informazione per uno specifico tag, il tag stesso potrebbe non esserci. Il vostro programma deve gestire queste possibilità.

Così quando gestiamo i dati XML, iniziamo con il root tag (ndt contrassegno di radice) e analizziamo ciascun tag – cercando i dati che ci interessano. In alcuni casi, vogliamo tutto; in altri ci interessiamo solo di certi pezzi di informazioni.

Ora, andiamo a dare una occhiata alla prima chiamata e vediamo che valore rende indietro. Assumente che lo spettacolo che stiamo cercando sia Buffy the Vampire Slayer. La nostra chiamata per la ricerca dovrebbe assomigliare a questa:

```
http://services.tvrage.com/feed
s/search.php?show=buffy
```

Il file XML restituito dovrebbe essere come questo: http://pastebin.com/Eh6ZtJ9N.

Notate che io ho inserito le indentazioni per rendervi più facile la lettura. Ora spezziamo il file XML per vedere che cosa abbiamo veramente.

<Results> Questa è la RADICE dei dati XML. L'ultima linea del flusso che riceviamo dovrebbe essere </Results>. Fondamentalmente questo segna l'inizio e la fine del flusso XML. Questo potrebbe essere di zero risultati o di 50

risultati.
<show> Questo è il nodo genitore che dice "quello che segue (fino al tag di chiusura di show) è la informazione relativa ad un singolo show televisivo". Anche questo è terminato dal suo tag finale </show> . Qualsiasi cosa tra questi due tag dovrebbe essere considerato come un pezzo di valore dell'informazione dello show.
<showid>2930</show> Questo è il tag showid. Questo contiene il sid che dobbiamo usare per ottenere le informazioni dello show. In questo caso 2930.
<name>Buffy the Vampire Slayer</name> Questo è il nome dello show.
<link>...</link> Questo sarebbe il collegamento allo show stesso (o nel caso di un episodio, le informazioni sull'episodio) sul sito di TVRage.
<country>...</country> Il paese di origine dello show.
…
</show>
</Results>

Nel caso del nostro programma, siamo davvero interessati in soli due campi <showid> e <name>. Possiamo anche considerare di fare attenzione anche al campo <started>. Questo perché raramente recuperemo un solo insieme di dati, specialmente se non diamo il nome completo dello spettacolo. Per esempio, se siamo interessati allo spettacolo "The Big Bang Theory", e lo abbiamo cercato usando solo la stringa "Big Bang" potremmo ricevere venti o più insiemi di dati perché qualsiasi cosa che avesse lontamente coinciso con "big" o "bang" sarebbe stato elencato. Comunque, se siamo interessati allo spettacolo "NCIS", e abbiamo fatto una ricerca, abbiamoricevuto indietro molte risposte. Alcune che non ci saremmo aspettati. Non solo avremmo ottenuto "NCIS", "NCIS Los Angeles", "the real NCIS", ma anche "The Streets of San Francisco" e "Da Vinci's Inquest" e molti altri, dato che le lettere "N" "C" "I" "S" sono in tutte queste, quasi in questo ordine.

Una volta che conosciamo lo showid che vogliamo, quindi possiamo richiedere le informazioni dello spettacolo per quel ID. I dati sono simili ai dati che abbiamo appena ricevuto indietro nella risposta della ricerca, ma più dettagliati. Dunque, usando Buffy come nella nostra richiesta di esempio, qui, (prossima pagina, destra) trovate una abbreviata versione del file XML.

Potete vedere che la maggior parte dei dati è incluso nel flusso di risposta alla ricerca originale. Comunque, cose come il canale, il paese del canale, la data e l'ora della messa in onda sono specifiche a questo insieme di risposta.

Poi avremmo richiesto l'elenco degli episodi. Se lo spettacolo ha una sola stagione e se ha o ha avuto solo sei episodi, questo flusso potrebbe essere corto. Comunque, prendiamo il caso di uno dei miei spettacoli televisivi preferiti Doctor Who. Doctor Who è uno spettacolo britannico che nella sua forma originale è iniziato nel 1963, ed è proseguito per 26 stagioni ('serie' per i nostri amici in UK) fino al 1989. La sua prima stagione da sola aveva 42 episodi, mentre le altre stagioni/serie avevano circa 24 episodi. Potete vedere come si potrebbe avere un ENORME flusso da analizzare.

Quello che otteniamo dalla nostra richiesta di elenco degli episodi è mostrato nella pagina seguente (ancora usando Buffy come esempio); sto per usare solo una parte del flusso in modo tale che voi abbiate una buona idea di che cosa viene restituito.

Così per ricapitolare, le informazioni di cui abbiamo davvero bisogno o desideriamo nella ricerca per showid del flusso del nome potrebbero essere

```
<showid>
<name>
<started>
```

Nel flusso di informazioni sullo spettacolo noi (normalmente) vogliamo...

```
<seasons>
<started>
<start date>
<origin_country>
<status>
<genres>
<runtime>
<network>
<airtime>
<airday>
<timezone>
```

e dal flusso della lista degli episodi...

```
<Season>
<episode number>
<season number>
```

```
<Showinfo>
   <showid>2930</showid>
   <showname>Buffy the Vampire Slayer</showname>
   <showlink>http://tvrage.com/Buffy_The_Vampire_Slayer</showlink>
   <seasons>7</seasons>
   <started>1997</started>
   <startdate>Mar/10/1997</startdate>
   <ended>May/20/2003</ended>
   <origin_country>US</origin_country>
   <status>Canceled/Ended</status>
   <classification>Scripted</classification>
   <genres>
      <genre>Action</genre>
      <genre>Adventure</genre>
      <genre>Comedy</genre>
      <genre>Drama</genre>
      <genre>Mystery</genre>
      <genre>Sci-Fi</genre>
   </genres>
   <runtime>60</runtime>
   <network  country="US">UPN</network>
   <airtime>20:00</airtime>
   <airday>Tuesday</airday>
   <timezone>GMT-5 -DST</timezone>
   <akas>
      <aka country="SE">Buffy &amp;amp; vampyrerna</aka>
      <aka country="DE">Buffy - Im Bann der Dämonen</aka>
      <aka country="NO">Buffy - Vampyrenes skrekk</aka>
      <aka country="HU">Buffy a vámpírok réme</aka>
      <aka country="FR">Buffy Contre les Vampires</aka>
      <aka country="IT">Buffy l'Ammazza Vampiri</aka>
      <aka country="PL">Buffy postrach wampirów</aka>
      <aka country="BR">Buffy, a Caça-Vampiros</aka>
      <aka country="PT">Buffy, a Caçadora de Vampiros</aka>
      <aka country="ES">Buffy, Cazavampiros</aka>
      <aka country="HR">Buffy, ubojica vampira</aka>
      <aka country="FI">Buffy, vampyyrintappaja</aka>
      <aka country="EE">Vampiiritapja Buffy</aka>
      <aka country="IS">Vampírubaninn Buffy</aka>
   </akas>
</Showinfo>
```

```
<production number>
<airdate>
<link>
<title>
```

Una parola sugli "avvisi". I dati del numero della stagione e del numero degli episodi non sono quello che voi potreste pensare come giusti. Nel caso dei dati da TVRage, il numero della stagione è il numero dell'episodio all'interno della stagione. Il numero dell'episodio è il numero per quell'episodio all'interno dell'intera vita della serie. Il numero di produzione è un numero che era usato internamente alla serie, che, per molte persone, significa poco se non niente.

Ora che abbiamo rinfrescato la nostra memoria sulle strutture dei file XML, e esaminato le chiamate API di TVRage, siamo pronti per iniziare a scrivere il nostro codice ma dobbiamo aspettare fino alla prossima volta.

Fino ad allora, godetevi le vacanze.

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.

```
<Show>
   <name>Buffy the Vampire Slayer</name>
   <totalseasons>7</totalseasons>
   <Episodelist>
      <Season no="1">
         <episode>
            <epnum>1</epnum>
            <seasonnum>01</seasonnum>
            <prodnum>4V01</prodnum>
            <airdate>1997-03-10</airdate>
            <link>http://www.tvrage.com/Buffy_The_Vampire_Slayer/episodes/28077</link>
            <title>Welcome to the Hellmouth (1)</title>
         </episode>
         <episode>
            <epnum>2</epnum>
            <seasonnum>02</seasonnum>
            <prodnum>4V02</prodnum>
            <airdate>1997-03-10</airdate>
            <link>http://www.tvrage.com/Buffy_The_Vampire_Slayer/episodes/28078</link>
            <title>The Harvest (2)</title>
         </episode>
         <episode>
            <epnum>3</epnum>
            <seasonnum>03</seasonnum>
            <prodnum>4V03</prodnum>
            <airdate>1997-03-17</airdate>
            <link>http://www.tvrage.com/Buffy_The_Vampire_Slayer/episodes/28079</link>
            <title>Witch</title>
         </episode>
         ...
      </Season>
   </Episodelist>
</Show>
```

Programmare In Python: Parte 40

Scritto da Greg Walters

La volta scorsa abbiamo avuto una discussione di massima sulle API di TVRag web. Ora inizieremo a guardare la scrittura del codice per lavorarci.

L'obiettivo di questa parte è di iniziare il processo di creazione del codice, che sarà un modulo riusabile che potrà essere importato in qualsiasi programma python e fornirà un facile accesso alle API.

Sebbene le API TVRage ci danno un certo numero di cose da poter fare, e molte di più la versione registrata, ci concentreremo solo su queste tre chiamate:
1 - Cercare la trasmissione tramite nome e ottenere lo ShowID
2 – Ottenere le informazioni sulla trasmissione in base allo ShowID
3 – Ottenere delle informazioni sullo specifico episodio in base allo ShowID

La volta scorsa vi ho mostrato le chiamate API "non registrate" e accessibili da chiunque. Questa volta useremo le chiamate registrate, utilizzando una mia chiave di registrazione. Sto per condividere con voi questa chiave (TVRage è a conoscenza del fatto). Comunque vi chiedo il favore, se userete le API, di registrarvi e ottenere la vostra chiave personale e di non abusare del sito. Vi prego inoltre di considerare di sostenere i loro continui sforzi facendo una donazione.

Creeremo tre routine principali per fare le chiamate e restituire le informazioni, tre routine che saranno usate per mostrare le informazioni (assumendo che stiamo in modalità "stand alone") e una routine Main per fare il lavoro, ancora supponendo che si stia lavorando in modalità "stand alone".

Ecco qui la lista delle routine che stiamo per creare (benché non tutte questa volta. Voglio lasciare spazio ad altro in questo numero).

```
def FindIdByName(self, showname, debug = 0)

def GetShowInfo(self, showid, debug = 0)

def GetEpisodeList(self, showid, debug = 0)

def DisplaySearchResult(self, ShowListDict)

def DisplayShowInfo(self, dict)

def DisplayEpisodeList(self, SeriesName, SeasonCount, EpisodeList)

def main()
```

La routine FindByName prende una stringa (showname), effettua la chiamata alle API, analizza la risposta XML e ritorna una lista di trasmissioni che coincidono con le informazioni in un dizionario, così questa sarà una lista di dizionari. GetShowInfo prende lo showid dalla routine precedente e ritorna un dizionario di informazioni relative alle serie. Anche GetEpisodeList usa lo showid dalla precedente routine e ritorna una lista di dizionari contenenti informazioni per ciascun episodio.

Useremo una serie di stringhe per tenere la chiave e l'URL di base per poi aggiungerci quello di cui abbiamo bisogno. Per esempio, considerate il seguente codice (lo svilupperemo più tardi).

```
self.ApiKey = "Itnl8IyY1hsR9n0IP6zI"

self.FindSeriesString = "http://services.tvrage.com/myfeeds/search.php?key="
```

La chiamata che abbiamo bisogno di inviare (per ottenere una lista di informazioni sulle serie con l'Id della serie) potrebbe essere:

http://services.tvrage.com/myfeeds/search.php?key=Itnl8IyY1hsR9n0IP6zI&show={ShowName}

Uniamo la stringa così...

```
strng = self.FindSeriesString + self.ApiKey + "&show=" + showname
```

Per gli scopi del test userò una trasmissione dal titolo "Continuum" che, se non l'avete mai vista, è un meraviglioso spettacolo di fantascienza sulla rete Showcase fuori dal Canada. Userò questa trasmissione per alcune ragioni. Primo, ci sono solo (al momento della scrittura) due trasmissioni che corrispondono alla stringa di ricerca "Continuum", ciò rende facile il debug, e in secondo luogo c'è attualmente solo una stagione di 10 episodi con cui avere a che fare.

Dovreste avere un' idea sul cosa si sta per cercare con le routine di analisi, così ho messo le chiamate con l'URL

completo qui sotto per poterle testare prima di iniziare con la codifica.

Cercare per nome della trasmissione... http://services.tvrage.com/myfeeds/search.php?key=Itnl8IyY1hsR9n0IP6zI&show=continuum

Recuperare le informazioni sulla serie usando lo ShowID (sid) http://services.tvrage.com/myfeeds/showinfo.php?key=Itnl8IyY1hsR9n0IP6zI&sid=30789

Recuperare la lista degli episodi e le informazioni usando lo showID (sid) http://services.tvrage.com/myfeeds/episode_list.php?key=Itnl8IyY1hsR9n0IP6zI&sid=30789

Ora che abbiamo ottenuto tutto questo, iniziamo con il nostro codice.

Creeremo un file dal nome "tvrage.py". Lo useremo per uno o due dei prossimi numeri.

Cominceremo con gli import mostrati in alto a destra.

Potete notare che stiamo per usare ElementTree per fare l'analisi del XML e urlib per le comunicazioni internet. La libreria sys è usata per sys.exit.

Imposteremo ora il ciclo principale in modo tale che possiamo testare le cose mano a mano che procediamo (in basso a destra). Ricordate che questa è l'ultima cosa nel nostro file sorgente.

Come detto prima, le prime quattro righe sono le nostre stringhe parziali per costruire l'URL per la funzione che vogliamo usare (GetEpisodeListString dovrebbe stare tutto su una linea). Le ultime quattro righe inizializzano le liste che useremo più tardi.

Primo impostiamo la stringa che sarà usata come URL (in mezzo a destra). Quindi impostiamo il socket con un timeout predefinito di 8 secondi. Poi effettuiamo la chiamata a urllib.urlopen con il nostro URL generato e riceveremo (speriamo) il nostro file XML nell'oggetto usock. Chiamiamo la configurazione per

```
#=============================================================
#                        IMPORTS
#=============================================================
from xml.etree import ElementTree as ET
import urllib
import sys
```

```
    def FindIdByName(self,showname,debug = 0):
        strng = self.FindSeriesString + self.ApiKey + "&show=" + showname
        urllib.socket.setdefaulttimeout(8)
        usock = urllib.urlopen(strng)
        tree = ET.parse(usock).getroot()
        usock.close()
        foundcounter = 0
        self.showlist = []
```

```
#=============================================================
#     Main loop
#=============================================================
if __name__ == "__main__":
    main()
```

Ora iniziamo la nostra classe. Il nome è "TvRage". Creeremo inoltre la routine __init__.

```
class TvRage:
    def __init__(self):
        self.ApiKey = "Itnl8IyY1hsR9n0IP6zI"
        self.FindSeriesString = "http://services.tvrage.com/myfeeds/search.php?key="
        self.GetShowInfoString = "http://services.tvrage.com/myfeeds/showinfo.php?key="
        self.GetEpisodeListString =
"http://services.tvrage.com/myfeeds/episode_list.php?key="
        self.ShowList = []
        self.ShowInfo = []
        self.EpisodeList = []
        self.EpisodeItem = []
```

ElementTree così da poter analizzare le informazioni contenute nel XML (se vi perdete qui, per favore rileggete i miei articoli sull'XML (parte 10, 11 e 12 che sono apparsi nei numeri 36,37,e 38 di FCM)). Poi chiudiamo il socket e inizializziamo il contatore per il numero di coincidenze trovate e ripristiniamo la lista showlist a una lista vuota.

Ora procederemo per passi attraverso le informazioni XML usando il tag 'show' come genitore per quello che vogliamo. Ricordate che le informazioni restituite sono piuttosto somiglianti a quanto mostrato in alto a destra.

Andremo in ogni gruppo di informazioni per il genitore 'show' analizzando le informazioni. In pratica, tutto quello di cui abbiamo realmente bisogno è il nome della trasmissione (<showname>) e lo showid (<showid>) mostrato in basso a sinistra, ma gestiremo tutti i risultati.

Discuterò il primo e voi capirete i rimanenti. Mano a mano che procederemo nelle informazioni, cercheremo i tag (in basso a sinistra) che coincidono con quelli voluti. Se ne troviamo qualcuno, gli assegniamo una variabile temporanea e la mettiamo dentro al dizionario come valore con chiave corrisponde a ciò che stiamo inserendo. Nel caso visto sopra, stiamo cercando il tag 'showid' nei dati XML. Quando lo troviamo, lo assegniamo come valore alla chiave 'ID' del dizionario.

La porzione successiva (nella prossima pagina in alto a destra) ha a che fare con il genere della trasmissione. Come potete vedere dal pezzettino di codice XML sopra, questa trasmissione ha quattro differenti generi in cui può essere catalogata. Azione, Crimine, Drammatico e Fantascienza. Dobbiamo gestire ciascuno di essi.

Infine, incrementiamo la variabile foundcounter e aggiungiamo questo

```
for node in tree.findall('show'):
    showinfo = []
    genrestring = None
    dict = {}
    for n in node:
        if n.tag == 'showid':
            showid = n.text
            dict['ID'] = showid
```

```
<Results>
    <show>
        <showid>30789</showid>
        <name>Continuum</name>
        <link>http://www.tvrage.com/Continuum</link>
        <country>CA</country>
        <started>2012</started>
        <ended>0</ended>
        <seasons>2</seasons>
        <status>Returning Series</status>
        <classification>Scripted</classification>
        <genres>
            <genre>Action</genre>
            <genre>Crime</genre>
            <genre>Drama</genre>
            <genre>Sci-Fi</genre>
        </genres>
    </show>
    ...
</Results>
```

```
    elif n.tag == 'name':
        showname = n.text
        dict['Name'] = showname
    elif n.tag == 'link':
        showlink = n.text
        dict['Link'] = showlink
    elif n.tag == 'country':
        showcountry = n.text
        dict['Country'] = showcountry
    elif n.tag == 'started':
        showstarted = n.text
        dict['Started'] = showstarted
    elif n.tag == 'ended':
        showended = n.text
        dict['Ended'] = showended
    elif n.tag == 'seasons':
        showseasons = n.text
        dict['Seasons'] = showseasons
    elif n.tag == 'status':
        showstatus = n.text
        dict['Status'] = showstatus
    elif n.tag == 'classification':
        showclassification = n.text
        dict['Classification'] = showclassification
```

dizionario nella lista 'showlist'. Quindi rifacciamo il procedimento fino a che non ci sono ulteriori dati XML. Una volta che tutto è fatto, restituiamo la lista dei dizionari (in basso a destra).

La maggior parte del codice è auto esplicativo. Ci concentreremo sul ciclo for, utilizzato per stampare le informazioni. Passiamo ciclicamente attraverso ogni elemento della lista dei dizionari e stampiamo una variabile contatore, il nome della trasmissione (c['Name']) e l'id. Il risultato assomiglia a questo ...

```
Enter Series Name -> continuum
2 Found
----------------------
1 - Continuum - 30789
2 - Continuum (Web series) -
32083
Enter Selection or 0 to exit ->
```

Per favore ricordate che la lista ha come base lo zero, così quando l'utente inserisce '1' sta in realtà chiedendo il dizionario numero 0. Facciamo questo perché 'normalmente' la gente pensa che il conteggio parta da '1' e non da zero. E quindi possiamo usare 0 per uscire dalla routine, evitando che usino 'Q' o 'q' o '-1'.

Ora la routine Main che collega il tutto per noi.

Per oggi, inizieremo appena la routine (in a mezzo a destra) e la continueremo la prossima volta.

La prossima volta aggiungeremo le altre routine. Per adesso il codice può essere trovato presso http://pastebin.com/6iw5NQrW

Arrivederci a presto.

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

```
elif n.tag == 'genres':
    for subelement in n:
        if subelement.tag == 'genre':
            if subelement.text != None:
                if genrestring == None:
                    genrestring = subelement.text
                else:
                    genrestring += " | " + subelement.text
    dict['Genres'] = genrestring
```

```
def main():
    tr = TvRage()
    #--------------------
    # Find Series by name
    #--------------------
    nam = raw_input("Enter Series Name -> ")
    if nam != None:
        sl = tr.FindIdByName(nam)
        which = tr.DisplayShowResult(sl)
        if which == 0:
            sys.exit()
        else:
            option = int(which)-1
            id = sl[option]['ID']
            print "ShowID selected was %s" % id
```

```
            foundcounter += 1
            self.showlist.append(dict)
        return self.showlist
#=======================================================
```

La prossima cosa che faremo sarà creare la routine che mostra tutti i nostri risultati.

```
    def DisplayShowResult(self, ShowListDict):
        lcnt = len(ShowListDict)
        print "%d Found" % lcnt
        print "----------------------"
        cntr = 1
        for c in ShowListDict:
            print "%d - %s - %s" % (cntr,c['Name'],c['ID'])
            cntr += 1
        sel = raw_input("Enter Selection or 0 to exit -> ")
        return sel
```

Programmare in Python: Parte 41

Lo scorso mese abbiamo iniziato la nostra versione a linea di comando di una libreria per parlare con le API di TVRage web. Questo mese continueremo a incrementare questa libreria. Se non avete il codice dal mese scorso, per favore prendetelo da pastebin (http://pastebin.com/6iw5NQrW) perché aggiungeremo a quel codice.

Nello stato in cui abbiamo lasciato il codice, voi potreste eseguire il programma e inserire nella finestra del terminale il nome di uno spettacolo televisivo di cui volete delle informazioni. Ricordate, abbiamo usato la trasmissione Continuum. Una volta premuto <Invio>, il programma chiamerà le API ed effettuerà una ricerca per nome dello spettacolo e quindi restituirà una lista di nomi che corrispondono alla vostra immissione. Dovete quindi scegliere dalla lista inserendo un numero e il programma dovrebbe mostrarvi "ShowID selected was 30789". Ora creeremo il codice che userà quello ShowID per ottenere informazioni sulla serie. Un'altra cosa da tenere in mente: le routine di visualizzazione esistono praticamente per provare che la routine funziona. L'obiettivo finale è creare una libreria riusabile che può essere reimpiegata in qualcosa come un programma con interfaccia utente grafica GUI. Sentitevi liberi di modificare le routine di visualizzazione se volete sfruttare di più le funzionalità autonome della libreria.

```
        def GetShowInfo(self,showid,debug=0):
            showidstr = str(showid)
            strng = self.GetShowInfoString + self.ApiKey + "&sid=" + showidstr
            urllib.socket.setdefaulttimeout(8)
            usock = urllib.urlopen(strng)
            tree = ET.parse(usock).getroot()
            usock.close()
            dict = {}
```

L'ultima routine che abbiamo creato nella classe era "DisplayShowResult". Appena dopo quella e prima della routine "main" ci metteremo la nostra successiva routine. Le informazioni che restituirà (ci sono altre informazioni, ma noi useremo solo quelle della lista qui sotto) saranno in un dizionario e conterranno (se disponibili):

- Show ID
- Show Name
- Show Link
- Origin Country of network
- Number of seasons
- Series image
- Year Started
- Date Started
- Date Ended
- Status (cancelled - cancellato, returning - sta per essere riproposto, current - in corso, etc)
- Classification (scripted - sceneggiato, reality, etc)
- Series Summary
- Genre(s)
- Runtime in minutes
- Name of the network that originally aired the show (chi lo ha messo in onda per primo)
- Network country (pressapoco la stessa cosa del Paese di Origine)
- Air time
- Air Day (della settimana)
- TimeZone

Quello mostrato sopra è l'inizio del codice.

Dovreste riconoscere la maggior

```
        for child in tree:
            if child.tag == 'showid':
                dict['ID'] = child.text
            elif child.tag == 'showname':
                dict['Name'] = child.text
            elif child.tag == 'showlink':
                dict['Link'] = child.text
            elif child.tag == 'origin_country':
                dict['Country'] = child.text
            elif child.tag == 'seasons':
                dict['Seasons'] = child.text
            elif child.tag == 'image':
                dict['Image'] = child.text
            elif child.tag == 'started':
                dict['Started'] = child.text
            elif child.tag == 'startdate':
                dict['StartDate'] = child.text
```

```
        elif child.tag == 'ended':
            dict['Ended'] = child.text
        elif child.tag == 'status':
            dict['Status'] = child.text
        elif child.tag == 'classification':
            dict['Classification'] = child.text
        elif child.tag == 'summary':
            dict['Summary'] = child.text
```

parte del codice dall'ultima volta. Non è cambiato molto. Qui c'è dell'altro codice (mostrato sotto).

Come potete vedere (sopra), non c'è niente di veramente nuovo anche in questo pezzo di codice, se state seguendo la serie di articoli. Stiamo usando un ciclo for per controllare ciascun tag nel file XML per uno specifico valore. Se lo troviamo, lo assegniamo a un elemento del dizionario.

Ora le cose diventano un po' più complicate. Controlleremo il tag "genres" (generi - N.d.T.). Questo ha dei tag figli al di sotto con il nome di "genre". Per ogni spettacolo fornito, ci possono essere molteplici generi. Dobbiamo appendere i generi a una stringa mano a mano che vengono su e separarli con un barra verticale e due spazi come questo " | " (mostrato in alto a destra).

Ora torniamo al codice "normale" (mostrato in mezzo a sinistra) che avete già visto: l'unica cosa un po' differente è il tag "network", che ha un attributo "country". Prendiamo i dati dell'attributo cercando "child.attrib['attributetag']" invece di "child.text".

Quella è la fine di questa routine. Ora (sotto) dobbiamo in qualche modo mostrare le informazioni ottenute per cui abbiamo lavorato duramente. Creeremo una routine chiamata "DisplayShowinfo".

Ora, dobbiamo aggiornare la routine "main" (nella prossima pagina, mostrata in alto a destra) per supportare le nostre due nuove routine. Vi sto dando l'intera routine qui sotto, ma il nuovo codice è mostrato **in neretto**.

```
elif child.tag == 'genres':
    genrestring = None
    for subelement in child:
        if subelement.tag == 'genre':
            if subelement.text != None:
                if genrestring == None:
                    genrestring = subelement.text
                else:
                    genrestring += " | " + subelement.text
    dict['Genres'] = genrestring
```

```
     elif child.tag == 'runtime':
         dict['Runtime'] = child.text
     elif child.tag == 'network': # has attribute
         dict['NetworkCountry'] = child.attrib['country']
         dict['Network'] = child.text
     elif child.tag == 'airtime':
         dict['Airtime'] = child.text
     elif child.tag == 'airday':
         dict['Airday'] = child.text
     elif child.tag == 'timezone':
         dict['Timezone'] = child.text
 return dict
```

Nella prossima pagina, in basso a sinistra, c'è il risultato prodotto da "DisplayShowInfo", così come dovrebbe apparire assumendo che abbiate scelto come spettacolo "Continuum".

Per favore notate che non sto mostrando le informazioni relative al fuso orario, ma sentitevi liberi di aggiungerle,

```
    def DisplayShowInfo(self,dict):
        print "Show: %s" % dict['Name']
        print "ID: %s  Started: %s  Ended: %s  Start Date: %s  Seasons: %s" %
(dict['ID'],dict['Started'],dict['Ended'],dict['StartDate'],dict['Seasons'])
        print "Link: %s" % dict['Link']
        print "Image: %s" % dict['Image']
        print "Country: %s  Status: %s   Classification: %s" %
(dict['Country'],dict['Status'],dict['Classification'])
        print "Runtime: %s  Network: %s   Airday: %s   Airtime: %s" %
(dict['Runtime'],dict['Network'],dict['Airday'],dict['Airtime'])
        print "Genres: %s" % dict['Genres']
        print "Summary: \n%s" % dict['Summary']
```

se preferite.

Poi abbiamo bisogno di lavorare sulle routine per la lista degli episodi della serie. La routine "che fa il lavoro" sarà chiamata "GetEpisodeList" e fornirà le seguenti informazioni …

- Season
- Episode Number
- Season Episode Number (il numero di episodi della stagione)
- Production Number
- Air Date
- Link
- Title
- Summary
- Rating
- Screen Capture Image of Episode (se disponibile)

Prima di partire con il codice, potrebbe essere utile rivisitare che cosa

```
def main():
    tr = TvRage()
    #--------------------
    # Find Series by name
    #--------------------
    nam = raw_input("Enter Series Name -> ")
    if nam != None:
        sl = tr.FindIdByName(nam)
        which = tr.DisplayShowResult(sl)
        if which == 0:
            sys.exit()
        else:
            option = int(which)-1
            id = sl[option]['ID']
            print "ShowID selected was %s" % id
    #-----------------------------
    # Get Show Info
    #-----------------------------
    showinfo = tr.GetShowInfo(id)
    #-----------------------------
    # Display Show Info
    #-----------------------------
    tr.DisplayShowInfo(showinfo)
```

```
ShowID selected was 30789
Show: Continuum
ID: 30789  Started: 2012  Ended: None  Start Date:
May/27/2012  Seasons: 2
Link: http://www.tvrage.com/Continuum
Image: http://images.tvrage.com/shows/31/30789.jpg
Country: CA  Status: Returning Series  Classification:
Scripted
Runtime: 60  Network: Showcase  Airday: Sunday
Airtime: 21:00
Genres: Action | Crime | Drama | Sci-Fi
Summary:
Continuum is a one-hour police drama centered on Kiera
Cameron, a regular cop from 65 years in the future who
finds herself trapped in present day Vancouver. She is
alone, a stranger in a strange land, and has eight of the
most ruthless criminals from the future, known as Liber8,
loose in the city.

Lucky for Kiera, through the use of her CMR (cellular
memory recall), a futuristic liquid chip technology
implanted in her brain, she connects with Alec Sadler, a
seventeen-year-old tech genius. When Kiera calls and Alec
answers, a very unique partnership begins.

Kiera's first desire is to get "home." But until she
figures out a way to do that, she must survive in our
time period and use all the resources available to her to
track and capture the terrorists before they alter
history enough to change the course of the future. After
all, what's the point of going back if the future isn't
the one you left?
```

restituisce la richiesta alle API della lista degli episodi. Somiglia a quello mostrato nella prossima pagina in alto a destra.

Le informazioni per ciascun episodio sono nel tag "episode", che è un figlio di "Season", che è un figlio di "EpisodeList", che é un figlio di "Show". Dobbiamo essere attenti a come analizziamo ciò.

Come la maggior parte delle routine che questa volta "fanno il lavoro", le prime linee (sotto) sono ora abbastanza facili da comprendere.

Dobbiamo quindi cercare i tag "name" e "totalseasons", sotto al tag "root" e al tag "Show". Una volta che abbiamo fatto con questi, cerchiamo i tag

```
        def GetEpisodeList(self,showid,debug=0):
            showidstr = str(showid)
            strng = self.GetEpisodeListString + self.ApiKey
+ "&sid=" + showidstr
            urllib.socket.setdefaulttimeout(8)
            usock = urllib.urlopen(strng)
            tree = ET.parse(usock).getroot()
            usock.close()
            for child in tree:
```

```
            if child.tag == 'name':
                ShowName = child.text
            elif child.tag == 'totalseasons':
                TotalSeasons = child.text
            elif child.tag == 'Episodelist':
                for c in child:
                    if c.tag == 'Season':
                        dict = {}
                        seasonnum = c.attrib['no']
                        for el in c:
```

"EpisodeList" e "Season". Notate che il tag "Season" ha un attributo. Potreste notare (nel suddetto codice) che non stiamo includendo nel dizionario i dati di "Showname"o "Totalseasons". Li stiamo assegnando a una variabile che sarà restituita, alla fine della routine, al codice chiamante.

Ora che abbiamo questa porzione di dati, diamoci da fare con le informazioni specifiche dell'episodio (mostrate sotto).

Quello che resta ora (in basso a sinistra) è appendere le informazioni specifiche dell'episodio (che abbiamo già messo nel dizionario) nella nostra lista e proseguire. Una volta che abbiamo fatto

```
<Show>
<name>Continuum</name>
<totalseasons>2</totalseasons>
<Episodelist>
<Season no="1">
<episode>
<epnum>1</epnum>
<seasonnum>01</seasonnum>
<prodnum/>
<airdate>2012-05-27</airdate>
<link>
http://www.tvrage.com/Continuum/episodes/1065162187
</link>
<title>A Stitch in Time</title>
<summary>
Inspector Kiera Cameron loses everything she has and finds
herself on a new mission when she and eight dangerous
terrorists are transported from their time in 2077 back to
2012 during the terrorist's attempt to escape execution.
She takes on a new identity and joins the VPD in order to
stop the terrorists' reign of violence. Along the way, she
befriends Alec Sadler, the 17 year old who will one day
grow up to create the technology her world is built upon.
</summary>
<rating>8.8</rating>
<screencap>
http://images.tvrage.com/screencaps/154/30789/1065162187.p
ng
</screencap>
</episode>
```

```
    if el.tag == 'episode':
        dict={}
        dict['Season'] = seasonnum
        for ep in el:
            if ep.tag == 'epnum':
                dict['EpisodeNumber'] = ep.text
            elif ep.tag == 'seasonnum':
                dict['SeasonEpisodeNumber'] = ep.text
            elif ep.tag == 'prodnum':
                dict['ProductionNumber'] = ep.text
            elif ep.tag == 'airdate':
                dict['AirDate'] = ep.text
            elif ep.tag == 'link':
                dict['Link'] = ep.text
            elif ep.tag == 'title':
                dict['Title'] = ep.text
            elif ep.tag == 'summary':
                dict['Summary'] = ep.text
            elif ep.tag == 'rating':
                dict['Rating'] = ep.text
            elif ep.tag == 'screencap':
                dict['ScreenCap'] = ep.text
```

con tutti gli episodi, ritorniamo alla routine chiamante e, come detto prima, restituiamo tre elementi di dati, "ShowName", "TotalSeasons" e la lista di dizionari.

Poi abbiamo bisogno di creare la nostra routine di visualizzazione. Di nuovo, è abbastanza facile. L'unica cosa che potreste non riconoscere sono le linee "if e.has_key('keynamehere')". Questo è un controllo per essere sicuri che ci siano veramente dei dati nelle variabili "Rating" e "Summary".

Alcuni spettacoli non hanno queste

```
self.EpisodeItem.append(dict)
        return ShowName,TotalSeasons,self.EpisodeItem
```

informazioni, così dobbiamo includere il controllo per rendere più carini i dati stampati a schermo (mostrati in alto a destra).

Tutto quello che rimane è aggiornare la nostra routine "main" (prossima pagina, in alto a destra). Ancora una volta, sto per fornirvi la routine "main" completa con il nuovo codice **in grassetto**.

Adesso, se salvate ed eseguite il programma, verranno visualizzati i risultati di "GetEpisodeList" e di "DisplayEpisodeList". Mostrato in basso a destra c'è un pezzetto delle informazioni sull'Episodio.

Questo è tutto per questo mese. Come al solito, potete trovare il codice sorgente completo su pastebin, presso http://pastebin.com/kWSEfs2E. Spero che vi divertiate a giocare con la libreria. Ci sono ulteriori informazioni disponibili dalle API che potete includere. Per favore ricordate, TVRage fornisce gratuitamente queste informazioni, quindi prendete in considerazione una donazione per aiutare il loro sforzo nell'aggiornamento delle API e per tutto il loro duro lavoro.

Arrivederci alla prossima volta. Divertitevi.

```
    def DisplayEpisodeList(self,SeriesName,SeasonCount,EpisodeList):
        print "-----------------------------------"
        print "Series Name: %s" % SeriesName
        print "Total number of seasons: %s" % SeasonCount
        print "Total number of episodes: %d" % len(EpisodeList)
        print "-----------------------------------"
        for e in EpisodeList:
            print "Season: %s" % e['Season']
            print "   Season Episode Number: %s - Series Episode Number: %s" %
(e['SeasonEpisodeNumber'],e['EpisodeNumber'])
            print "   Title: %s" % e['Title']
            if e.has_key('Rating'):
                print "   Airdate: %s   Rating: %s" % (e['AirDate'],e['Rating'])
            else:
                print "   Airdate: %s   Rating: NONE" % e['AirDate']
            if e.has_key('Summary'):
                print "   Summary: \n%s" % e['Summary']
            else:
                print "   Summary: NA"
            print "============================"
        print "------------ End of episode list ------------"
```

```
-----------------------------------
Series Name: Continuum
Total number of seasons: 2
Total number of episodes: 10
-----------------------------------
Season: 1
   Season Episode Number: 01 - Series Episode Number: 1
   Title: A Stitch in Time
   Airdate: 2012-05-27   Rating: 8.8
   Summary:
Inspector Kiera Cameron loses everything she has and finds herself on a new mission when
she and eight dangerous terrorists are transported from their time in 2077 back to 2012
during the terrorist's attempt to escape execution. She takes on a new identity and
joins the VPD in order to stop the terrorists' reign of violence. Along the way, she
befriends Alec Sadler, the 17 year old who will one day grow up to create the technology
her world is built upon.
============================
```

```
def main():
    tr = TvRage()
    #--------------------
    # Find Series by name
    #--------------------
    nam = raw_input("Enter Series Name -> ")
    if nam != None:
        sl = tr.FindIdByName(nam)
        which = tr.DisplayShowResult(sl)
        if which == 0:
            sys.exit()
        else:
            option = int(which)-1
            id = sl[option]['ID']
            print "ShowID selected was %s" % id
    #-----------------------------
    # Get Show Info
    #-----------------------------
    showinfo = tr.GetShowInfo(id)
    #-----------------------------
    # Display Show Info
    #-----------------------------
    tr.DisplayShowInfo(showinfo)
    #-----------------------------
    # Get Episode List
    #-----------------------------
    SeriesName,TotalSeasons,episodelist = tr.GetEpisodeList(id)
    #-----------------------------
    # Display Episode List
    #-----------------------------
    tr.DisplayEpisodeList(SeriesName,TotalSeasons,episodelist)
    #-----------------------------
```

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.

Programmare In Python: Parte 42

Assumiamo che voi abbiate deciso di creare un centro multimediale per la vostra stanza di famiglia. Avete un computer dedicato per il meraviglioso programma XMBC. Avete speso dei giorni per estrarre i vostri film in DVD e le serie TV sul computer: avete fatto la ricerca e nominato i file nel modo corretto. Ma diciamo che una delle vostre serie preferite sia NCIS e voi avete in DVD ogni episodio che potete ottenere. Avete trovato un posto che fornisce l'episodio corrente. Voi volete trovare quale sia il prossimo episodio e quando sarà trasmesso. In più volete una lista di tutti gli episodi in TV per impressionare i vostri amici.

Questo è il progetto che stiamo per iniziare questo mese. Il nostro primo compito è scavare nella cartella che contiene i vostri spettacoli TV, estrarre il nome della serie includendo il nome e il numero della stagione e il numero dell'episodio. Tutte questa informazioni andranno in un database per un facile immagazzinamento.

Secondo XMBC, dovreste denominare ciascuno dei vostri file di episodio in questo modo:

```
Tv.Show.Name.SxxExx.Episode
name here if you care.extension
```

Così,lasciateci usare il primo episodio di NCIS come esempio. Il nome del file per un file AVI dovrebbe essere:

```
NCIS.S01E01.Yankee White.avi
```

e l'ultimo episodio sarebbe:

```
NCIS.S10E17.Prime Suspect.avi
```

Se voi avete il nome di uno spettacolo che ha più di una parola, potrebbe assomigliare a questo:

```
Doctor.Who.2005.S07E04.The
Power of Three.mp4
```

La struttura della directory dovrebbe essere come la seguente:

```
TVShows
      2 Broke Girls
            Season 1
            Episode 1
            Episode 2
            ...
            Season 2
            ...
      Doctor Who 2005
            Season 1
            ...
            Season 2
            ...
```

e così via. Ora che sappiamo che cosa stiamo cercando e dove sarà, procediamo.

Molto tempo fa, creammo un programma per fare un database dei nostri file MP3. Credo sia stato nel numero 35 e che fosse parte del numero 9 di questa serie. Usammo una routine WalkThePath per cercare ricorsivamente attraverso tutte le cartelle di un percorso iniziale e tirare fuori tutti i nomi di file che avevano una estensione ".mp3". Riutilizzeremo molto di quella routine e la modificheremo per i nostri scopi. In questa versione stiamo cercando i file video che hanno una di queste estensioni:

```
.avi
.mkv
.m4v
.mp4
```

che sono estensioni molto comuni per i file video nel mondo dei media del pc.

Ora cominceremo con la prima parte del nostro progetto: create un file chiamato "tvfilesearch.py". Assicuratevi di salvarlo quando avremo finito questo mese perché ci lavoreremo sopra il mese prossimo.

Cominciamo con i nostri import:

```
import os
from os.path import join,
getsize, exists
import sys
import apsw
import re
```

Come potete vedere stiamo importando le librerie os, sys e apsw. Le abbiamo già usate tutte in precedenza. Stiamo inoltre importando la libreria 're' per supportare le Espressioni Regolari. Toccheremo rapidamente questo argomento questa volta, ma lo faremo in modo più approfondito la prossima volta.

Ora cominciamo a fare le nostre routine finali (pagina successiva). Tutto il nostro ulteriore codice andrà tra gli import e queste due routine finali.

Questa (pagina successiva, in alto a destra) è la nostra principale routine

di lavoro. In questa noi creiamo la connessione al database SQLite fornito da apsw. Quindi creiamo un cursore per interagire con questa. Quindi chiamiamo la routine MakeDatabase che creerà il database se non esiste.

I miei file TV sono situati su due dischi rigidi. Così ho creato una lista per contenere i nomi dei percorsi. Se voi avete una sola posizione di archiviazione, potete cambiare le tre linee per renderle come le seguenti:

```
    startfolder =
"/filepath/folder/"
    WalkThePath(startfolder)
```

Poi creiamo la nostra routine standard " if name"

```
#==========================
if __name__ == '__main__':
    main()
```

Ora che tutte che le cose poco interessanti sono state fatte, possiamo muoverci verso la parte succosa del nostro progetto. Cominceremo con la routine MakeDatabase (al centro). Mettetela giusto dopo gli import.

Discutemmo questa routine prima quando trattammo lo scanner MP3, così vi rammento solo che in questa routine abbiamo dei controlli per vedere se la tabella esiste, altrimenti, se non esiste, la creiamo.

Ora creeremo la routine "WalkThePath" (la seconda dal basso)

Quando noi inseriamo la routine,

```
#=========================================
def main():
    global connection
    global cursor
    # Create the connection and cursor.
    connection = apsw.Connection("TvShows.db3")
    cursor = connection.cursor()
    MakeDataBase()
```

```
#=========================================
def MakeDataBase():
    # IF the table does not exist, this will create the table.
    # Otherwise, this will be ignored due to the 'IF NOT EXISTS' clause
    sql = 'CREATE TABLE IF NOT EXISTS TvShows (pkID INTEGER PRIMARY KEY, Series TEXT,
RootPath TEXT, Filename TEXT, Season TEXT, EPISODE TEXT);'
    cursor.execute(sql)
```

forniamo il percorso dei file all'interno dei quali stiamo cercando. Puliamo la variabile showname, che useremo più tardi e apriamo un file per i log di errore. Quindi lasciamo fare alla routine le sue cose. Riceviamo di ritorno dalla chiamata (os.walk) una 3-tupla(percorso della directory, nomi delle directory, nomi dei file). Il percorso della directory è una stringa che è il percorso per la directory, i nomi della directory è una lista dei nomi delle subdirectory nel percorso e i nomi dei file è una lista dei nomi che non sono directory. Controlliamo quindi attraverso la lista dei nomi dei file controllando per vedere se il nome del file termina con una delle nostre estensioni obbiettivo.

```
#=========================================
def WalkThePath(filepath):
    showname = ""
    # Open the error log file
    efile = open('errors.log',"w")
    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'))]:
```

```
    #=========================================
    # Set your video media paths
    #=========================================
    startfolder = ["/extramedia/tv_files/","/media/freeagnt/tv_files_2/"]
    for cntr in range(0,2):
        WalkThePath(startfolder[cntr])
        # Close the cursor and the database
    cursor.close()
    connection.close()
    print("Finished")
```

Ora suddividiamo il nome completo del file nella estensione e nel nome del file (senza estensione). Poi chiamiamo la routine GetSeasonEpisode per tirare fuori le informazioni Stagione/Episodio che sono dentro al nome del file assumendo che siano correttamente formattate.

```
OriginalFilename,ext =
os.path.splitext(file)

fl = file

isok,data =
GetSeasonEpisode(fl)
```

GetSeasonEpisode ritorna un boolean e una lista (in questo caso "data") che contiene il nome della serie,la stagione, e i numeri degli episodi. Se un nome di file non ha il formato corretto, la variabile booleana "isok" (in alto a destra) sarà false.

Poi (al centro), controlleremo per vedere se il file sia nel database. Se è così, non vogliamo duplicarlo. Semplicemente controlliamo usando il nome del file. Possiamo andare più in profondità e assicurarsi che il percorso sia lo stesso ma per adesso questo è sufficiente.

Se tutto funziona come dovrebbe, la risposta dalla query dovrebbe essere o 1 o 0. Se è 0 allora non c'è e scriveremo l'informazione nel database. Se c'è andiamo oltre. Notate i comandi Try Except sopra e sotto. Se qualcosa va male, come qualche carattere che il database non gradisce, evitiamo che il programma abortisca. Teniamo comunque traccia nel log dell'errore in modo tale che possiamo gestirlo più tardi.

Stiamo semplicemente inserendo un nuovo record nel database o scrivendo nel file degli errori.

```
                    if isok:
                showname = data[0]
                season = data[1]
                episode = data[2]
                print("Season {0} Episode {1}".format(season,episode))
            else:
                print("No Season/EPisode")
                efile.writelines('--------------------------\n')
                efile.writelines('{0} has no series/episode information\n'.format(file))
                efile.writelines('--------------------------\n\n')
```

```
            sqlquery = 'SELECT count(pkid) as rowcount from TvShows where Filename =
"%s";' % fl
            try:
                for x in cursor.execute(sqlquery):
                    rcntr = x[0]
                if rcntr == 0:  # It's not there, so add it
```

```
                    try:
                        sql = 'INSERT INTO TvShows (Series,RootPath,Filename,Season,Episode)
VALUES (?,?,?,?,?)'
                        cursor.execute(sql,(showname,root,fl,season,episode))
                    except:
                        print("Error")
                        efile.writelines('--------------------------\n')
                        efile.writelines('Error writing to database...\n')
                        efile.writelines('Filename = {0}\n'.format(file))
                        efile.writelines('--------------------------\n\n')
                except:
                    print("Error")
                print('Series - {0} File - {1}'.format(showname,file))
```

```
    # Close the log file
        efile.close
# End of WalkThePath
```

Ora diamo una occhiata alla routine GetSeasonEpisode.

```
#==============================
==========
def GetSeasonEpisode(filename):
    filename = filename.upper()
    resp =
re.search(r'(.*).S\d\dE\d\d(\.*
)', filename, re.M|re.I)
```

La porzione re.search del codice è parte della re library. Usa una stringa come modello e, in questo caso, il nome del file che vogliamo analizzare. I re.M|re.I sono i parametri che dicono che vogliamo usare una ricerca di tipo multilinea (re.M) combinato con un ignora maiuscolo minuscolo (re.I). Come ho detto prima approfondiremo le espressioni regolari il prossimo mese, dal momento che la nostra routine un solo tipo di stringa series|episode. Per il modello di ricerca stiamo cercando per: ".S" seguito da due numeri decimali , seguito da una "E" in maiuscolo, quindi due ulteriori cifre e quindi un punto. Se il nostro nome di file è come "tvshow.S01E03.avi", questo troverà una corrispondenza. Comunque alcune persone codificheranno i loro show in modi tipo "tvshow.s01s03.avi"o "tvshow.103.avi", diventa così molto più difficile da gestire. Modificheremo questa routine il prossimo mese per coprire la maggioranza della casistiche. La "r" permette di usare una stringa libera per una ricerca.

Proseguendo, la ricerca ritorna un oggetto in cui noi possiamo guardare. "resp" è una risposta che è vuota se non vi è alcuna corrispondenza e, in questo caso, vengono restituiti due gruppi di informazioni. Il primo ci darà i caratteri fino alla corrispondenza, mentre il secondo include la corrispondenza. Così nel caso descritto sopra, group(1) dovrebbe essere "tvshow", il secondo gruppo dovrebbe essere "tvshow.S01E03". Questo è specificato dalle parentesi nella ricerca "(.*)" e "(\.*)".

```
    if resp:
        showname =
resp.group(1)
```

Prendiamo il nome dello show dal gruppo numero uno: poi prendiamo la lunghezza di questo in modo tale che possiamo estrarre la stringa con la serie e l'episodio con un comando substring.

```
        shownamelength =
len(showname) + 1
        se =
filename[shownamelength:shownam
elength+6]
        season = se[1:3]
        episode = se[4:6]
```

Poi sostituiamo tutti i punti nel nome dello show con uno spazio per essere più "leggibile da un umano".

```
        showname =
showname.replace("."," ")
```

Creiamo una lista per includere il nome dello spettacolo, stagione ed episodio e per ritornarlo con il booleano valorizzato a True per dire che le cose sono andate bene.

```
        ret =
[showname,season,episode]
        return True,ret
```

Altrimenti se non troviamo una corrispondenza, creiamo la nostra lista contenete nessun nome dello show e due numeri -1, questo farà ritornare un valore booleano False.

```
    else:
        ret = ["",-1,-1]
        return False,ret
```

Questo è tutto il codice. Adesso andiamo a vedere come dovrebbe apparire il risultato in uscita. Dando per scontato che le strutture dei file siano esattamente come le mie, alcuni dei risultati sullo schermo dovrebbe apparire come questo ...

```
Season 02 Episode 04
SELECT count(pkid) as rowcount
from TvShows where Filename =
"InSecurity.S02E04.avi";
Series - INSECURITY File -
InSecurity.S02E04.avi
Season 01 Episode 08
SELECT count(pkid) as rowcount
from TvShows where Filename =
"Prime.Suspect.US.S01E08.Underw
ater.avi";
Series - PRIME SUSPECT US File
-
Prime.Suspect.US.S01E08.Underwa
ter.avi
```

e così via. Potete accorciare il risultato in uscita per impedire allo schermo di farvi diventare pazzi se lo preferite. Come abbiamo detto prima, ciascun elemento che troviamo viene messo dentro al database. Qualcosa come questo:

```
pkID | Series | Root Path
     | Filename
     | Season | Episode
2526 | NCIS   |
/extramedia/tv_files/NCIS/Seaso
n
7|NCIS.S07E04.Good.Cop.Bad.Cop.
avi | 7     | 4
```

Come sempre il listato dell'intero codice è disponibile su PasteBin.com a http://pastebin.com/txmmagkL

La prossima volta parleremo ulteriormente dei formati Season|Episode, e alcune altre cose per approfondire il nostro programma.

Arrivederci a presto.

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

Programmare in Python: Parte 43

Scritto da Greg Walters

La volta scorsa, abbiamo iniziato un progetto che alla fine userà il modulo TvRage che abbiamo creato il mese prima di questo. Ora continueremo il progetto. Questa volta aggiungeremo funzionalità al nostro programma: migliorando la routine che analizza il nome del file e aggiungendo due campi (TvRageId e Status) al database. Quindi, tuffiamoci dentro.

Come primo passo, faremo dei cambiamenti alle nostre linee di importazione: Per quelli che si stanno unendo adesso, includerò quelle della volta scorsa (mostrate in alto a destra).

Le linee dopo ' import re' sono quelle nuove per questa volta.

La prossima cosa che faremo è riscrivere la routine GetSeasonEpisode. Stiamo per buttare via quasi tutto ciò che abbiamo fatto il mese scorso e renderlo più flessibile attraverso i possibili schemi stagione/episodio . In questa iterazione saremo in grado di supportare i seguenti schemi ...

```
Series.S00E00

Series.s00e00

Series.S00E00.S00E01

Series.00x00

Series.S0000

Series.0x00
```

Sistemeremo inoltre i problemi legati a eventuali zero iniziali mancanti prima di scrivere nel database.

Il nostro primo modello cerca di prendere file con episodi multipli. Ci sono vari nomi di schemi, ma quello che supporteremo è simile a 'S01E03.S01E04'. Usiamo come modello la stringa "(.*)\.s(\d{1,2})e(\d{1,2})\.s(\d{1,2})e(\d{1,2})". Questo ritorna (speriamo) cinque gruppi composti da: nome della serie (S[1]), stagione(S[2]), episodio numero 1 (S[3]), stagione (S[4]) ed episodio numero 2 (S[5]). Ricordate che le parentesi creano ciascun gruppo per il ritorno. Nel caso sopra, raggruppiamo dal primo carattere fino a ".s", quindi due numeri, saltiamo la "e", quindi due numeri, e ripetiamo. Così il nome del file "Monk.S01E05.S01E06.avi" ritorna i seguenti gruppi ...

```
import os
from os.path import join, getsize, exists
import sys
import apsw
import re
#-----------------------------
#   NEW LINES START HERE
#-----------------------------
from xml.etree import ElementTree as ET
import urllib
import string
from TvRage import TvRage
```

```
S[1] = Monk

S[2] = 01

S[3] = 05

S[4] = 01

S[5] = 06
```

Stiamo usando solo i gruppi S[1], S[2] e S[3] in questo codice, ma si può vedere a cosa conducono. Se troviamo una corrispondenza, impostiamo una variabile di nome "GoOn" a vero. Questo ci permette di sapere che cosa fare dopo che siamo passati attraverso le varie linee If.

Così, nella prossima pagina (in alto a destra) c'è il codice per la routine GetSeasonEpisode.

Quando arriviamo a questo punto, (prossima pagina, in basso a sinistra) prepariamo il nome dello spettacolo rimuovendone tutti i punti dal nome, inserendo quindi le informazioni sulla stagione e sull'episodio dai vari gruppi e le restituiamo. Per le informazioni sulla stagione, se abbiamo un modello tipo "S00E00", la stagione avrà uno zero iniziale. Tuttavia, se il modello è simile a "xxx", allora la stagione si presume essere il primo carattere e i successivi due sono l'episodio. Per far capire dove vogliamo andare a parare, vogliamo rendere la stagione un numero a due cifre con uno zero iniziale, se necessario.

Successivamente, nella nostra routine MakeDatabase, cambieremo l'istruzione SQL di creazione per

aggiungere i due nuovi campi (prossima pagina, in alto)

Di nuovo, l'unica cosa che è cambiata dalla scorsa volta sono le ultime due definizioni di campi.

Nella nostra funzione WalkThePath, gli unici cambiamenti sono le linee che inseriscono effettivamente nel database. Questo è per supportare le nuove strutture. Se vi ricordate dalla volta scorsa, passiamo a questa funzione la cartella che contiene i nostri file TV. Nel mio caso, ci sono due cartelle, così è impostato dentro una lista e usiamo un ciclo for per passare ciascun elemento nella funzione. Procedendo attraverso la funzione, andiamo in ogni directory cercando i file con estensione del tipo .avi, .mkv, .mp4 e .m4v. Quando troviamo un file che corrisponde, lo mandiamo alla

```
def GetSeasonEpisode(filename):
    GoOn = False
    filename = filename.upper()
```

This is our first pattern check.

```
    #Should catch multi episode .S01E01.S01E02 type filenames
    resp = re.search(r'(.*)\.s(\d{1,2})e(\d{1,2})\.s(\d{1,2})e(\d{1,2})',filename, re.I)
    if resp:
        showname = resp.group(1)
        GoOn = True
    else:
```

Our second pattern check looks for SddEdd or sddedd...

```
    # Should catch SddEdd or sddedd
        resp = re.search(r'(.*).S(\d\d?)E(\d\d?)(\.*)', filename, re.I)
        if resp:
            showname = resp.group(1)
            GoOn = True
        else:
```

The next pattern looks for ddxdd.

```
       #check for ddxdd
       resp = re.search(r'(.*)\.(\d{1,2})x(\d{1,2})(.*)', filename, re.I)
       if resp:
           showname = resp.group(1)
           GoOn = True
       else:
```

This pattern checks for Sdddd.

```
                #check for Sdddd
                resp = re.search(r'(.*).S(\d\d)(.\d\d?)' , filename, re.I)
            if resp:
                showname = resp.group(1)
                GoOn = True
            else:
```

And finally we try for ddd

```
     # Should catch xxx
     resp = re.search(r'(.*)(\d)(.\d\d?)',filename,re.I)
     if resp:
         showname = resp.group(1)
         GoOn = True
```

```
if GoOn:
    shownamelength = len(showname) + 1
    showname = showname.replace("."," ")
    season = resp.group(2)
    if len(season) == 1:
        season = "0" + season
    episode = resp.group(3)
    ret = [showname,season,episode]
    return True,ret
else:
    ret = ["",-1,-1]
    return False,ret
```

```
def MakeDataBase():
    # IF the table does not exist, this will create the table.
    # Otherwise, this will be ignored due to the 'IF NOT EXISTS' clause
    sql = 'CREATE TABLE IF NOT EXISTS TvShows (pkID INTEGER PRIMARY KEY, Series TEXT, RootPath TEXT, Filename TEXT,
Season TEXT, Episode TEXT, tvrageid TEXT,status TEXT);'
    cursor.execute(sql)
```

funzione GetSeasonEpisode. Verifichiamo quindi se lo abbiamo già inserito nel database e, se non fatto, lo aggiungiamo. Sto per darvi (in alto a destra) solo parte della routine dello scorso mese.

Le due linee in nero sono quelle nuove di questa volta.

Abbiamo già fatto metà strada. Ci sono poi delle funzioni di supporto per lavorare con la nostra funzione TvRage per riempire i campi del database. La nostra prima funzione viene eseguita dopo quella WalkThe Path e accede al database, ottenendo il nome della serie e interrogando il server TvRage per il numero identificativo. Una volta ottenuto, aggiorniamo il database, usiamo quindi tale numero id per interrogare ancora una volta il server TvRage per ottenere lo stato attuale della serie. Questo stato può essere “New Series”, Returning Series”, “Canceled”, “Ended” e “On Haitus”. La motivazione per cui vogliamo questa informazione è che, quando andiamo a controllare i

```
            sqlquery = 'SELECT count(pkid) as rowcount from TvShows where Filename =
"%s";' % fl
            try:
                for x in cursor.execute(sqlquery):
                    rcntr = x[0]
                if rcntr == 0:  # It's not there, so add it
                    try:
                        sql = 'INSERT INTO TvShows
(Series,RootPath,Filename,Season,Episode,tvrageid) VALUES (?,?,?,?,?,?)'
                        cursor.execute(sql,(showname,root,fl,season,episode,-1))
                    except:
```

```
def WalkTheDatabase():
    tr = TvRage()
    SeriesCursor = connection.cursor()
    sqlstring = "SELECT DISTINCT series FROM TvShows WHERE tvrageid = -1"
```

nuovi episodi, non vogliamo perdere tempo con delle serie che non avranno altri nuovi episodi perché sono state cancellate. Così ora abbiamo lo stato e possiamo scriverlo nel database (sopra).

Ci fermeremo qui col nostro codice e daremo una occhiata alla query SQL che stiamo usando. E' un pochino differente da tutto ciò che abbiamo fatto prima. La stringa è:

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

Che dice, dammi giusto una istanza del nome della serie, non importa quante ne ho, dove il campo tvrageid è uguale a "-1". Se, per esempio, abbiamo 103 episodi di Doctor Who 2005, usando il Distinct otterrò solo un record, assumendo che non abbiamo ancora ricevuto un TvRageID.

```
    for x in
SeriesCursor.execute(sqlstring)
:
        seriesname = x[0]

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

Stiamo usando la funzione capwords dalla libreria string per cambiare il nome della serie (x[0]) al “corretto caso” dal formato tutto maiuscolo con cui di solito salviamo il nome dello spettacolo. Facciamo questo perché TvRage si aspetta qualcosa di diverso dai dati tutti in maiuscolo e non otterremmo i risultati che stiamo cercando. Così il nome

```
def UpdateDatabase(seriesname,id):
    idcursor = connection.cursor()
    sqlstring = 'UPDATE tvshows SET tvrageid = ' + id + ' WHERE series = "' + seriesname + '"'
    try:
        idcursor.execute(sqlstring)
    except:
        print "error"
```

```
def GetShowStatus(seriesname,id):
    tr = TvRage()
    idcursor = connection.cursor()
    dict = tr.GetShowInfo(id)
    status = dict['Status']
    sqlstring = 'UPDATE tvshows SET status = "' + status + '" WHERE series = "' + seriesname + '"'
    try:
        idcursor.execute(sqlstring)
    except:
        print "Error"
```

della serie "THE MAN FROM UNCLE" sarà convertito in "The Man From Uncle". Lo usiamo nella chiamata alla nostra libreria Libreria FindIdByName di TvRage. Questo ci fornisce l'elenco di spettacoli che corrispondono e ce li mostra in modo da scegliere il migliore. Una volta che ne prendiamo uno, aggiorniamo il database con il numero id e quindi chiamiamo la funzione GetShowStatus per ottenere lo stato attuale dello spettacolo da TvRage (in basso a destra).

La funzione UpdateDatabase (in alto) usa semplicemente il nome della serie come chiave per aggiornare tutti record con l'appropriato ID TvRage.

Anche GetShowStatus (sopra) è molto semplice. Chiamiamo la funzione GetShowInfo dalla libreria TvRage passando l'id che abbiamo appena ricevuto a TvRage, per ottenere le informazioni sulla serie. Se ricordate, TvRage fornisce molte informazioni sulla serie, ma tutto quello che ci interessa adesso è lo stato dello spettacolo. Dal momento che tutto è ritornato in un dizionario, cerchiamo la chiave ["Status"]. Una volta ottenuta, aggiorniamo il database con tale chiave e procediamo.

```
    print("Requesting information on " + searchname)
    sl = tr.FindIdByName(searchname)
    which = tr.DisplayShowResult(sl)
    if which == 0:
        print("Nothing found for %s" % seriesname)
    else:
        option = int(which)-1
        id = sl[option]['ID']
        UpdateDatabase(seriesname,id)
        GetShowStatus(seriesname,id)
```

Abbiamo quasi finito con il nostro codice. Alla fine aggiungiamo una linea alla nostra funzione main del mese

```
startfolder = ["/extramedia/tv_files","/media/freeagnt/tv_files_2"]
#for cntr in range(0,2):
    #WalkThePath(startfolder[cntr])
WalkTheDatabase()
# Close the cursor and the database
cursor.close()
connection.close()
print("Finished")
```

scorso (in nero, sotto) per chiamare la funzione "WalkTheDatabase" dopo che abbiamo finito di ottenere tutti i nostri nomi di file. Di nuovo, sto per darvi solo parte della funzione Main, giusto perché possiate trovare il posto corretto per piazzare la nuova linea.

Questo è tutto il nostro codice. Andiamo mentalmente oltre a quello che succede quando mandiamo in esecuzione il programma.

Primo, creiamo il database se non esiste.

Quindi, passiamo attraverso i percorsi predefiniti, cercando i file che hanno una delle seguenti estensioni:

```
.AVI, .MKV, .M4V, .MP4
```

Quando ne troviamo uno, lo scorriamo e cerchiamo di analizzare il nome del file cercando il nome di una serie, il numero di stagione e il numero di episodio. Prendiamo questa informazione e la mettiamo nel database, se non c'è ancora.

Una volta che abbiamo cercato i file, interroghiamo il database cercando i nomi delle serie che non hanno un ID TvRage associato. Poi interrogheremo le API TvRage e chiederemo i file che corrispondono a quell'ID. Ciascuna serie passerà attraverso questa fase ancora una volta. La seguente schermata mostra, in questo caso, le opzioni per la serie tv Midsomer Murders.

Ho inserito (in questo caso) 1, che associa queste serie con l'ID TvRage 4466. Questo è inserito nel database e usato in seguito per richiedere lo stato attuale della serie, di nuovo da TvRage. In questo caso ci viene restituito "Returning Serie", che viene quindi inserito nel database e proseguiamo.

Dal momento che facciamo l'esecuzione iniziale nel database, ci vorrà un certo tempo e richiederà la vostra attenzione, perché ogni singola serie ha bisogno di chiedere che il numero ID corrisponda. La buona notizia è che ciò deve essere fatto una sola volta. Se siete in qualche modo "normali" non vorrete aver a che fare con ciò. Io ho 157 differenti serie da fare, quindi ciò richiede un certo tempo. Poiché sono stato attento nell'impostare i nomi dei file (controllando TvRage e TheTvDb.com per la corretta denominazione del nome della serie), la maggioranza delle ricerche erano l'opzione #1.

Giusto per conoscenza, oltre la metà delle serie che ho o sono finite o

```
Requesting information on Midsomer Murders
5 Found
------------------------
1 - Midsomer Murders - 4466
2 - Motives and Murders - 31373
3 - See No Evil: The Moors Murders - 11199
4 - The Atlanta Child Murders - 26402
5 - Motives & Murders: Cracking the Case - 33322
Enter Selection or 0 to exit ->
```

sono state cancellate. Questo dovrebbe dirvi qualcosa sul gruppo di età in cui ricado.

Il codice completo è disponibile, come sempre, su PasteBin presso http://pastebin.com/MeuGyKpX

La prossima volta continueremo con l'integrazione con TvRage. Fino ad allora vi auguro di trascorrere un buon mese!

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.