

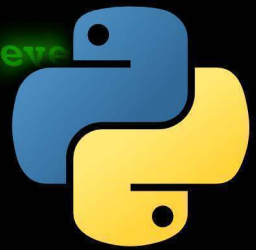


full circle

LA RIVISTA INDIPENDENTE PER LA COMUNITÀ LINUX UBUNTU
EDIZIONE SPECIALE SERIE PROGRAMMAZIONE



EDIZIONE SPECIALE
SERIE PROGRAMMAZIONE



python

PROGRAMMARE IN PYTHON VOLUME 2



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.

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 9-16**, pubblicata nei numeri 35-42: niente di speciale, giusto quello che abbiamo già pubblicato.

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

Buon divertimento!

Come contattarci

Sito web:

<http://www.fullcirclemagazine.org/>

Forum:

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

IRC:

#fullcirclemagazine su
chat.freenode.net

Gruppo editoriale

Capo redattore: Ronnie Tucker

(aka: RonnieTucker)

ronnie@fullcirclemagazine.org

Webmaster: Rob Kerfia

(aka: admin / linuxgeekery-)

admin@fullcirclemagazine.org

Podcaster: Robin Catling

(aka RobinCatling)

podcast@fullcirclemagazine.org

Manager delle comunicazioni:

Robert Clipsham

(aka: mrmonday) -

mrmonday@fullcirclemagazine.org



Gli articoli contenuti in questa rivista sono stati rilasciati sotto la licenza Creative Commons Attribuzione - Non commerciale - Condividi allo stesso modo 3.0. Ciò significa che potete adattare, copiare, distribuire e inviare gli articoli ma solo sotto le seguenti condizioni: dovete attribuire il lavoro all'autore originale in una qualche forma (almeno un nome, un'email o un indirizzo Internet) e a questa rivista col suo nome ("Full Circle Magazine") e con suo indirizzo Internet www.fullcirclemagazine.org (ma non attribuire il/gli articolo/i in alcun modo che lasci intendere che gli autori e la rivista abbiano esplicitamente autorizzato voi o l'uso che fate dell'opera). Se alterate, trasformate o create un'opera su questo lavoro dovete distribuire il lavoro risultante con la stessa licenza o una simile o compatibile.

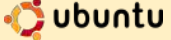
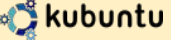
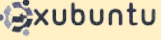
Full Circle magazine è completamente indipendente da Canonical, lo sponsor dei progetti di Ubuntu, e i punti di vista e le opinioni espresse nella rivista non sono in alcun modo da attribuire o approvati da Canonical.



VEDI ANCHE:

FCM#27-34 - Python Parti 1 - 8

VALIDO PER:

CATEGORIE:

    
Sviluppo Grafica Internet M/media Sistema

DISPOSITIVI:

    
CD/DVD HDD USB Drive Laptop Wireless

Se somigliate a me, avrete nei vostri computer una parte della vostra musica preferita in formato mp3. Quando si hanno meno di 1.000 file è piuttosto facile ricordare cosa si possiede e dove. Io, d'altra parte, ne ho molti di più. In una vita precedente ero un DJ e ho convertito la maggior parte della mia musica diversi anni fa. Il problema principale che dovetti affrontare fu lo spazio su

disco. Ora il problema più grande è ricordare cosa ho e dove si trova.

In questa e nella prossima lezione vedremo di creare un catalogo per i nostri MP3. Ci soffermeremo anche su alcuni nuovi concetti python nel rivisitare le nostre conoscenze dei database.

Primo, un file MP3 può conservare informazioni su se stesso. Il titolo della canzone, l'album, l'artista e altre informazioni. Queste sono contenute nei tag ID3 e sono riferite come metadati. All'inizio le informazioni che potevano essere conservate erano limitate. Originariamente erano conservate alla fine del file in un blocco di 128 byte. A causa delle dimensioni di questo blocco era possibile salvare 30 caratteri per il titolo, il nome dell'artista, e così via. Per molti file musicali era sufficiente ma (e questa è una delle mie canzoni preferite di sempre) quando ne avete una con il

titolo "Clowns (The Demise of the European Circus with No Thanks to Fellini)", potete salvare solo i primi 30 caratteri. Questo era frustrante per molti. Così, lo "standard" ID3 divenne noto come ID3v1 e fu creato un nuovo formato chiamato, abbastanza incredibilmente, ID3v2. Il nuovo formato permetteva di conservare informazioni variabili in lunghezza all'inizio del file, mentre il vecchio ID3v1 restava alla fine così da tornare utile per i lettori più vecchi. Oggi il contenitore per metadati può contenere fino a 256MB. L'ideale per le stazioni radio e per i pazzoidi come me. Nell'ID3v2 ciascun gruppo di informazioni è contenuto in frame e ciascun frame ha un identificativo. In una versione iniziale di ID3v2 l'identificativo era lungo tre caratteri. La versione odierna (ID3v2.4) ne usa uno di quattro.

All'inizio era possibile aprire il file in modalità binaria e cercare le informazioni che

volevamo, ma richiedeva molto lavoro perché non erano disponibili librerie standard. Oggi abbiamo numerose librerie che lo fanno al posto nostro. Per il nostro progetto ne useremo una chiamata Mutagen. Dovrete andare in Synaptic e installare python-mutagen. Se volete, potete fare una ricerca per "ID3". Vi accorgete che ci sono 90 pacchetti (in karmic), e se digitate "Python" nel campo di ricerca rapida troverete 8 pacchetti. Ci sono pro e contro per ognuno di essi ma, per il nostro progetto, proveremo Mutagen. Sentitevi liberi di provare gli altri per estendere le vostre conoscenze.

Ora che avete installato Mutagen, inizieremo a scrivere il codice.

Avviate un nuovo progetto e chiamatelo "mCat". Inizieremo con i vari import.

```
from mutagen.mp3 import MP3
```



```
import os

from os.path import
join, getsize, exists

import sys

import apsw
```

Per la maggior parte li avete visti in precedenza. Quindi, vogliamo creare le intestazioni delle nostre funzioni.

```
def MakeDataBase():
    pass
def S2HMS(t):
    pass
def WalkThePath(musicpath):
    pass
def error(message):
    pass
def main():
    pass
def usage():
    pass
```

Ahhh...una cosa nuova. Ora abbiamo una funzione main e una funzione usage. A cosa servono? Aggiungiamo qualcos'altro prima di discuterne.

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

Cosa diavolo è questo? Sì

tratta di un trucco per usare il nostro file sia come un programma a se stante sia come modulo riutilizzabile importato in un'altra applicazione. In pratica dice "SE questo file è il programma principale, dovremo eseguire la routine main altrimenti lo useremo come un modulo di supporto e le funzioni saranno chiamate direttamente da un altro programma".

Quindi, arricchiremo la funzione usage. Sotto c'è l'intero codice per la routine.

Qui creeremo un messaggio da mostrare all'utente che avvierà il nostro programma senza un parametro necessario per permetterne un uso

standalone. Notate che usiamo '\n' per forzare una nuova riga e '\t' per forzare una tabulazione. Usiamo anche '%' per includere il nome dell'applicazione contenuto in sys.argv[0]. Quindi usiamo la funzione error per mostrare il messaggio, poi si esce dall'applicazione (sys.exit(1)).

Continuiamo con la routine error. Ecco la versione completa.

```
def error(message):
    print >> sys.stderr,
    str(message)
```

Qui abbiamo usato la cosiddetta redirezione (">>"). Quando usiamo la funzione "print", stiamo dicendo a python

che vogliamo visualizzare l'output, o lo stream, al dispositivo standard di output, in genere il terminale in uso. Per fare questo usiamo (in modo trasparente) stdout. Quando vogliamo inviare un messaggio di errore, useremo lo stream stderr. Anche questo è il terminale. Quindi reindirizziamo l'output allo stream stderr.

Ora lavoriamo sulla routine main. Qui imposteremo la nostra connessione e il cursore per il nostro database, quindi controlleremo i nostri argomenti parametri di sistema e, se tutto è corretto, chiameremo la nostra funzione che eseguirà il compito assegnatole. Ecco il codice:

```
def usage():
    message = (
        '=====\n'
        'mCat - Finds all *.mp3 files in a given folder (and sub-folders),\n'
        '\tread the id3 tags, and write that information to a SQLite database.\n\n'
        'Usage:\n'
        '\t{0} <foldername>\n'
        '\t WHERE <foldername> is the path to your MP3 files.\n\n'
        'Author: Greg Walters\n'
        'For Full Circle Magazine\n'
        '=====\n'
    ).format(sys.argv[0])
    error(message)
    sys.exit(1)
```

```
def main():
    global connection
    global cursor
    #-----
    if len(sys.argv) != 2:
        usage()
    else:
        StartFolder = sys.argv[1]
        if not exists(StartFolder): # From os.path
            print('Path {0} does not seem to
            exist...Exiting.').format(StartFolder)
            sys.exit(1)
        else:
            print('About to work {0}
            folder(s)').format(StartFolder)
            # Create the connection and cursor.
            connection=apsw.Connection("mCat.db3")
            cursor=connection.cursor()
            # Make the database if it doesn't exist...
            MakeDataBase()
            # Do the actual work...
            WalkThePath(StartFolder)
            # Close the cursor and connection...
            cursor.close()
            connection.close()
            # Let us know we are finished...
            print("FINISHED!")
```

Come abbiamo fatto la volta precedente, creiamo due variabili globali chiamate connection e cursor per il nostro database. Quindi controlliamo i parametri (se ce ne sono) passati dalla riga di comando nel terminale. Per questo ricorriamo al comando sys.argv. Stiamo cercando due parametri, prima il

nome dell'applicazione, che è automatico, e quindi il percorso dei nostri file MP3. Se i due parametri mancano, salteremo alla routine usage, che stampa il messaggio a schermo e poi esce. In questo caso ricadiamo nella clausola else della nostra istruzione IF. Successivamente, inseriamo il parametro del percorso iniziale nella variabile

StartFolder. Attenzione che se nel percorso è presente uno spazio, per esempio, (/mnt/musicmain/Adult Contemporary) i caratteri dopo di esso saranno visti come un altro parametro. Così quando usate un percorso con uno spazio assicuratevi di includerlo tra virgolette. Quindi impostiamo la connessione e il cursore, creiamo il database e poi eseguiamo il lavoro vero e proprio nella routine WalkThePath e per finire chiudiamo il nostro cursore e la connessione al database e diciamo all'utente che abbiamo finito. L'intera funzione WalkThePath può essere trovata qui:

<http://pastebin.com/CegsAXjW>.

Prima cancelliamo i tre contatori usati per tenere traccia del lavoro fatto. Quindi apriamo un file che conterrà il registro degli errori nel caso ve ne sia bisogno. Quindi percorriamo ricorsivamente il percorso indicato dall'utente. In pratica, iniziamo dal percorso fornito ed entriamo e usciamo dalle sottocartelle presenti, alla ricerca dei file con estensione

".mp3". Quindi incrementiamo il contatore della cartella e poi quello dei file per tenere traccia del numero dei file processati. Quindi passeremo in rassegna ciascun file. Cancelliamo la variabile locale che contiene le informazioni di ciascuna canzone. Usiamo la funzione join da os.path per creare un percorso e un nome di file appropriati così da poter dire a mutagen dove trovare il file. Poi passiamo il suo nome alla classe MP3 ricevendone una istanza di "audio". Quindi recuperiamo tutti i tag ID3 che il file contiene e scorriamo la lista alla ricerca dei tag che ci interessano e li assegnamo alle nostre variabili temporanee. In questo modo si riducono al minimo gli errori. Date un'occhiata al codice che si occupa del numero della traccia. Quando mutagen lo restituisce può essere un valore singolo, un valore come "4/18" o come _trk[0] e _trk[1] o può essere vuoto. Usiamo il costrutto try/except per catturare gli eventuali errori. Ora controllate il codice che salva i dati nel database. Stiamo procedendo diversamente rispetto all'ultima volta. Creiamo l'istruzione SQL

come prima, ma questa volta sostituiamo il valore delle variabili con "?". Quindi inseriamo i valori nell'istruzione `cursor.execute`. Secondo il sito ASPW questo è il sistema migliore per il nostro caso, quindi lo prendo per oro colato. Per finire ci occupiamo della gestione di altri eventuali errori cui si potrebbe incorrere. Per la maggior parte si tratta di `TypeError` o `ValueError` generati eventualmente da caratteri Unicode ingestibili. Date un'occhiata alla strana maniera con cui formattiamo e mostriamo le stringhe. Non usiamo il carattere di sostituzione '%'. Usiamo la sostituzione con "{0}" che fa parte delle specifiche di Python 3.x. La forma base è:

```
Print('String that will be
printed with {0} number of
statements").format(replacemen
t values)
```

Per `efile.writelines` usiamo pure la sintassi classica.

Per finire dovremo controllare la routine `S2HMS`. La funzione prende la durata della canzone recuperata da `mutagen` ed

espressa tramite un float e la converte in una stringa nel formato "Ore:Minuti:Secondi" oppure "Minuti:Secondi". Guardate le istruzioni `return`. Ancora una volta, stiamo usando la sintassi Python 3.x. Comunque, c'è qualcosa di nuovo in pentola. Stiamo usando tre serie di sostituzioni (0, 1 e 2), ma cosa sono i ":02n" dopo i numeri 1 e 2? Dice che vogliamo, se necessari, degli zero iniziali. Così se una canzone dura 2 minuti e 4 secondi, la stringa restituita sarà "2:04", non "2:4".

L'intero codice del nostro programma lo trovate qui: <http://pastebin.com/rFf4Gm7E>.

Scandagliate la rete e vedete cosa potete trovare su `Mutagen`. Il suo uso va oltre gli MP3.



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.

LA MIA STORIA VELOCEMENTE

Il mio studio è completamente digitale con quattro macchine con Windows XP in una rete peer to peer. La quinta macchina esegue Linux Ubuntu 9.04 esclusivamente come sistema di test per Linux. Ho iniziato con Ubuntu 7.04 ed ho aggiornato ad ogni nuova uscita. L'ho trovato molto stabile, facile da usare e configurare dato che ogni versione migliora il SO.

In questo momento è solo il mio banco di prova ma è collegato alla mia rete e condivide dati con le macchine con Windows. Sono molto soddisfatto della stabilità di Ubuntu per gli aggiornamenti, i programmi, il supporto hardware e l'aggiornamento dei driver. Anche se è un peccato che i produttori più importanti come Adobe non prevedano una versione dedicata, Wine sembra funzionare bene. Ci sono programmi grafici e stampanti professionali attinenti alla mia attrezzatura fotografica che non funzionano così devo aspettare che Wine migliori o che il software venga portato.

Audio, video, CD/DVD, USB e unità Zip tutte sembrano funzionare appena collegate, il che è fantastico. Alcuni difetti nel software ci sono ma sembrano di minore importanza.

Tutto sommato Ubuntu è visivamente fresco e divertente da usare. Non sono un secchione così non uso la riga di comando a meno che non venga incuriosito da un tutorial e non decida di provarlo; la GUI del SO è abbastanza completa per noi non-secchioni che vogliamo usare solo l'interfaccia grafica.

Scarico Full Circle Magazine tutti i mesi e lo condivido con uno dei miei colleghi per mostrargli cosa è disponibile. Molte persone ancora non conoscono questo SO e come è facile da usare, ma lo scontento che provoca Microsoft lo farà crescere di più. L'unica cosa che assolutamente amo di questo SO è la capacità di chiudere un programma dal comportamento anomalo. Il pulsante per l'interruzione funziona prontamente ed elimina la frustrazione di aspettare che Windows XP si sblocchi. Perché Windows non può fare una cosa altrettanto semplice? Raramente però ho bisogno di usare il pulsante, a dimostrazione di quanto sia stabile Linux.

Brian G Hartnell - *Fotografo*



VEDI ANCHE:

FCM nn. 27-35 - Python Parti 1-9

VALIDO PER:

CATEGORIE:



DISPOSITIVI:



Language (linguaggio marcatore estensibile), analogamente all'HTML. Fu progettato per conservare e trasferire dati efficacemente attraverso Internet o altre vie di comunicazione. XML è essenzialmente un file di testo formattato usando propri tag e dovrebbe essere abbastanza auto-documentante. Essendo un file di testo, può essere compresso per un trasferimento più rapido e facile. Rispetto all'HTML, XML non fa nulla da sè. Non si cura di come i vostri dati devono apparire. Come detto poco fa, XML non obbliga ad attenersi a tag standard. Potete crearne di vostri.

Diamo un'occhiata a un generico file XML:

```
<root>
  <node1>Data Here</node1>
  <node2
attribute="something">Node 2
data</node2>
  <node3>
    <node3sub1>more
data</node3sub1>
  </node3>
</root>
```

La prima cosa da notare è l'indentazione. In realtà, l'indentazione si usa solo per comodità. Il file XML funzionerebbe anche se scritto così...

```
<root><node1>Data
Here</node1><node2
attribute="something">Node 2
data</node2><node3><node3sub1
>more
data</node3sub1></node3></root>
```

Continuando, i tag contenuti nelle parentesi "<>" devono rispettare alcune regole. Primo, devono essere una parola unica. Secondo, quando avete un tag iniziale (per esempio <root>) dovete avere un corrispondente tag di chiusura. Questo tag inizia con "/". I tag sono sensibili alle maiuscole: <node>, <Node>, <NODE> e <NodE> sono tutti tag diversi, e quelli di chiusura devono corrispondere. I nomi dei tag possono contenere lettere, numeri e altri caratteri, ma non possono iniziare con un numero o caratteri di punteggiatura. Dovreste evitare "-", ".", e ":"

perché alcuni software li potrebbero considerare comandi o proprietà di un oggetto. Inoltre, i due punti sono riservati per altro. Ci si riferisce ai tag come elementi.

Ogni file XML è essenzialmente un albero - inizia da una radice e da lì si suddivide. Ogni file XML DEVE avere una radice, che è il genitore di ogni altra cosa nel file. Tornate al nostro esempio. Dopo la radice, ci sono tre elementi figli: node1, node2 e node3. Mentre tutti sono figli dell'elemento root, node3 è genitore di node3sub1.

Ora osservate node2. Notate come all'interno delle parentesi vi sia oltre al solito dato anche qualcosa chiamato attribute. Oggigiorno, molti sviluppatori evitano gli attributi poiché gli elementi sono altrettanto efficaci e danno meno problemi, ma scoprirete che gli attributi vengono ancora utilizzati. Li approfondiremo a breve.

Probabilmente avete già sentito il termine XML. Potreste, però, non sapere cosa significa. XML sarà l'oggetto della lezione di questo mese. Gli obiettivi sono:

- Familiarizzare con il concetto di XML
- Leggere e scrivere file XML tramite i vostri programmi
- Prepararvi per un successivo più importante progetto XML

Così... parliamo di XML. XML sta per EXTensible Markup

Diamo un'occhiata all'utile esempio in basso.

Qui abbiamo l'elemento radice chiamato "people" che contiene due elementi figli chiamati "person". Ciascun 'person' ha 6 elementi figli: `firstname`, `lastname`, `gender`, `address`, `city` e `state`. A prima vista potreste pensare a questo file XML come a un database (ricordando le ultime lezioni), e avreste ragione. Infatti, alcune applicazioni usano i file XML come semplici strutture database. Ora, scrivere un programma per leggere un file XML potrebbe essere fatto senza tanti problemi.

```
<people>
  <person>
    <firstname>Samantha</firstname>
    <lastname>Pharoh</lastname>
    <gender>Female</gender>
    <address>123 Main St.</address>
    <city>Denver</city>
    <state>Colorado</state>
  </person>
  <person>
    <firstname>Steve</firstname>
    <lastname>Levon</lastname>
    <gender>Male</gender>
    <address>332120 Arapahoe Blvd.</address>
    <city>Denver</city>
    <state>Colorado</state>
  </person>
</people>
```

Semplicemente, aprire il file, leggere ciascuna riga e, basandosi sull'elemento, trattare il dato e quindi chiudere il file quando si è finito. Però ci sono sistemi migliori.

Negli esempi che seguono, useremo il modulo di libreria chiamato `ElementTree`. Potete recuperarlo direttamente con Synaptic installando `python-elementtree`. Tuttavia io ho scelto di visitare il sito di `ElementTree` (<http://effbot.org/downloads/#elementtree>) e scaricare direttamente il file sorgente (`elementtree-1.2.6-20050316.tar.gz`). Una volta

scaricato ho usato il gestore di pacchetti per estrarne il contenuto in una cartella temporanea. Sono entrato nella cartella e ho eseguito "sudo `python setup.py install`". Questo ha posizionato i file nella cartella `common` di python così posso usarlo sia in python 2.5 che 2.6. Ora possiamo iniziare a lavorare. Create una cartella per il codice di questo mese, copiate il codice XML di sopra nel vostro editor di testo preferito e salvatelo nella nuova cartella come "xmlsample1.xml".

Ora il nostro codice. La prima cosa che vogliamo fare è testare la nostra installazione di `ElementTree`. ecco il codice:

```
import
elementtree.Element
tree as ET

tree =
ET.parse('xmlsampl
el.xml')

ET.dump(tree)
```

Quando eseguiamo il programma di test, dovremmo avere qualcosa di simile a quello mostrato in basso a destra.

Tutto quello che abbiamo fatto è stato permettere a `ElementTree` di aprire il file, analizzarlo nelle sue parti base ed estrarlo così come è in memoria. Poca fantasia in questo caso.

Ora sostituite il vostro codice

```
/usr/bin/python -u
"/home/greg/Documents/articles/xml/rea
der1.py"

<people>
  <person>
    <firstname>Samantha</firstname>
    <lastname>Pharoh</lastname>
    <gender>Female</gender>
    <address>123 Main St.</address>
    <city>Denver</city>
    <state>Colorado</state>
  </person>
  <person>
    <firstname>Steve</firstname>
    <lastname>Levon</lastname>
    <gender>Male</gender>
    <address>332120 Arapahoe
Blvd.</address>
    <city>Denver</city>
    <state>Colorado</state>
  </person>
</people>
```


con il seguente:

```
import
elementtree.ElementTree as ET

tree =
ET.parse('xmlsample1.xml')

person =
tree.findall('..//person')

for p in person:
    for dat in p:
        print "Element: %s -
Data: %s" %(dat.tag,dat.text)
```

ed eseguitelo di nuovo. Ora il vostro output dovrebbe essere:

```
/usr/bin/python -u
"/home/greg/Documents/articles
/xml/reader1.py"
```

```
Element: firstname - Data:
Samantha
Element: lastname - Data:
Pharoh
Element: gender - Data: Female
Element: address - Data: 123
Main St.
Element: city - Data: Denver
Element: state - Data:
Colorado
Element: firstname - Data:
Steve
Element: lastname - Data:
Levon
Element: gender - Data: Male
Element: address - Data:
332120 Arapahoe Blvd.
Element: city - Data: Denver
Element: state - Data:
Colorado
```

Quindi ciascun dato è accompagnato dal nome del tag. Per capire con cosa abbiamo a che fare potremmo fare una semplice stampa. Analizziamo che cosa abbiamo realizzato. Abbiamo fatto analizzare il file a ElementTree che ha messo il risultato nell'oggetto tree. Quindi abbiamo chiesto a ElementTree di trovare tutte le occorrenze di person. Nell'esempio fatto ce ne sono due ma potrebbero essere 1 o 1000. Person è figlio di people e noi sappiamo che quest'ultimo è semplicemente la radice. Tutti i nostri dati sono ripartiti in person. Quindi abbiamo creato un semplice

ciclo for per analizzare ciascun oggetto person. Abbiamo poi creato un altro ciclo for per estrarre i dati in ciascun person e visualizzarli mostrando il nome dell'elemento (.tag) e il dato (.text).

Ora un esempio più reale. La mia famiglia ed io ci divertiamo in un'attività chiamata Geocaching. Se non sapete cosa sia sappiate che si tratta di una caccia al tesoro per "secchioni" in cui si usa un dispositivo GPS portatile per trovare ciò che un altro ha nascosto. Si pubblicano le coordinate GPS grezze in un sito web, qualche volta con indizi, si inseriscono le coordinate nel nostro GPS e

quindi inizia la caccia. Secondo Wikipedia ci sono oltre 1.000.000 di siti attivi in tutto il mondo, quindi probabilmente ce ne sono alcuni nella vostra zona. Io ne uso due per ottenere località in cui cercare. Uno è <http://www.geocaching.com/> e l'altro è <http://navicache.com/>. Ce ne sono altri, ma questi due sono quelli più importanti.

In genere i siti di geocaching usano file XML per le loro informazioni. Ci sono applicazioni che recuperano questi dati e li trasferiscono al dispositivo GPS. Alcune funzionano da database, permettendovi di tenere traccia della vostra attività, talvolta con

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<loc version="1.0" src="NaviCache">
  <waypoint>
    <name id="N02CAC"><![CDATA[Take Goofy Pictures at Grapevine Lake by g_phillips
Open Cache: Unrestricted
Cache Type: Normal
Cache Size: Normal
Difficulty: 1.5
Terrain : 2.0]]></name>
    <coord lat="32.9890166666667" lon="-97.0728833333333" />
    <type>Geocache</type>
    <link text="Cache Details">http://www.navicache.com/cgi-
bin/db/displaycache2.pl?CacheID=11436</link>
  </waypoint>
</loc>
```

Navicache file

mappe. Per il momento ci concentreremo nell'analisi dei file scaricati.

Sono andato su Navicache e ho trovato un nascondiglio decente in Texas. Il contenuto del file è mostrato nella pagina precedente.

Copiate i dati dal box e salvateli come "Cache.loc". Prima di iniziare a programmare, esaminiamo il file cache.

La prima riga essenzialmente ci informa che si tratta di un file XML convalidato, quindi possiamo tranquillamente ignorarla. La riga successiva (quella che inizia con "loc") è la nostra radice e contiene gli attributi "version" e "src". Ricordate, ho già detto che alcuni file utilizzano gli attributi. Ne avremo a che fare con altri proseguendo nel file. Di nuovo, la radice in questo caso può essere ignorata. La riga successiva fornisce il figlio waypoint (un waypoint è il luogo dove, in questo caso, si trova la cache). Ora recuperiamo i dati che ci interessano davvero. C'è il nome della cache, le

coordinate in latitudine e longitudine, il tipo di cache e un collegamento alla pagina web con ulteriori informazioni.

L'elemento name è una stringa lunga contenente diverse informazioni utili ma che dobbiamo analizzare noi stessi. Ora creiamo una nuova applicazione per leggere e mostrare il file. Chiamiamola "readacache.py". Iniziamo dall'import e dal parse dell'esempio precedente.

```
import
elementtree.ElementTree as ET
tree = ET.parse('Cache.loc')
```

Adesso vogliamo recuperare solo i dati all'interno del tag waypoint. Per farlo usiamo la funzione .find di ElementTree. Mettiamo il risultato nell'oggetto "w".

```
w = tree.find('./waypoint')
```

Quindi, dobbiamo analizzare tutti i dati. Per farlo useremo un ciclo loop. All'interno del ciclo, controlliamo il tag per cercare gli elementi 'name', 'coord', 'type' e 'link'. In base al tag che otteniamo, estrarremo l'informazione per stamparla in

seguito.

```
for w1 in w:
    if w1.tag == "name":
```

Poiché il tag 'name' sarà il primo che controlleremo, rivediamo i dati che otterremo.

```
<name
id="N02CAC"><![CDATA[Take
Goofy Pictures at Grapevine
Lake by g_phillips
```

```
Open Cache: Unrestricted
```

```
Cache Type: Normal
```

```
Cache Size: Normal
```

```
Difficulty: 1.5
```

```
Terrain : 2.0]]></name>
```

È davvero una stringa molto lunga. L'id della cache è

```
# Get text of cache name up to the phrase "Open Cache: "
CacheName = w1.text[:w1.text.find("Open Cache: ")-1]
# Get the text between "Open Cache: " and "Cache Type: "
OpenCache = w1.text[w1.text.find("Open Cache: ")
+12:w1.text.find("Cache Type: ")-1]
# More of the same
CacheType = w1.text[w1.text.find("Cache Type: ")
+12:w1.text.find("Cache Size: ")-1]
CacheSize = w1.text[w1.text.find("Cache Size: ")
+12:w1.text.find("Difficulty: ")-1]
Difficulty= w1.text[w1.text.find("Difficulty: ")
+12:w1.text.find("Terrain : ")-1]
Terrain = w1.text[w1.text.find("Terrain :")+12:]
```

impostato come attributo. Il nome è la parte dopo "CDATA" e prima di "Open Cache:". Andremo a frammentare la stringa in parti più piccole. Per ricavare una parte della stringa usiamo:

```
newstring =
oldstring[startposition:endpos
ition]
```

Così possiamo usare il codice sotto per recuperare l'informazione necessaria.

Quindi dobbiamo recuperare l'id localizzato nell'attributo del tag name. Controlliamo se ci sono attributi (cosa che sappiamo vera), così:

```
if w1.keys():
    for name,value in
```

PROGRAMMARE IN PYTHON - PARTE 10

```
wl.items():
    if name == 'id':
        CacheID = value
```

Ora possiamo passare al codice per le coordinate, il tipo e il link mostrato in basso a destra. In conclusione, per visionarli li stamperemo usando il codice in fondo a destra. Più a destra c'è il codice completo.

Avete imparato abbastanza per leggere la maggior parte dei file XML. Come sempre, potete recuperare l'intero codice di questa lezione sul mio sito web <http://www.thedesignatedgeek.com>.

La prossima volta useremo le nostre conoscenze XML per raccogliere informazioni da un meraviglioso sito meteo e mostrarle nel terminale. Buon divertimento!



Greg Walter 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.

```
elif wl.tag == "coord":
    if wl.keys():
        for name,value in wl.items():
            if name == "lat":
                Lat = value
            elif name == "lon":
                Lon = value
elif wl.tag == "type":
    GType = wl.text
elif wl.tag == "link":
    if wl.keys():
        for name, value in wl.items():
            Info = value
    Link = wl.text
```

```
print "Cache Name: ",CacheName
print "Cache ID: ",CacheID
print "Open Cache: ",OpenCache
print "Cache Type: ",CacheType
print "Cache Size: ",CacheSize
print "Difficulty: ", Difficulty
print "Terrain: ",Terrain
print "Lat: ",Lat
print "Lon: ",Lon
print "GType: ",GType
print "Link: ",Link
```

```
import elementtree.ElementTree as ET
tree = ET.parse('Cache.loc')
w = tree.find('./waypoint')
for wl in w:
    if wl.tag == "name":
        # Get text of cache name up to the phrase "Open Cache:
        ..
        CacheName = wl.text[:wl.text.find("Open Cache: ")-1]
        # Get the text between "Open Cache: " and "Cache Type:
        ..
        OpenCache = wl.text[wl.text.find("Open Cache:
")+12:wl.text.find("Cache Type: ")-1]
        # More of the same
        CacheType = wl.text[wl.text.find("Cache Type:
")+12:wl.text.find("Cache Size: ")-1]
        CacheSize = wl.text[wl.text.find("Cache Size:
")+12:wl.text.find("Difficulty: ")-1]
        Difficulty= wl.text[wl.text.find("Difficulty:
")+12:wl.text.find("Terrain  : ")-1]
        Terrain = wl.text[wl.text.find("Terrain  :")+12:]
        if wl.keys():
            for name,value in wl.items():
                if name == 'id':
                    CacheID = value
    elif wl.tag == "coord":
        if wl.keys():
            for name,value in wl.items():
                if name == "lat":
                    Lat = value
                elif name == "lon":
                    Lon = value
    elif wl.tag == "type":
        GType = wl.text
    elif wl.tag == "link":
        if wl.keys():
            for name, value in wl.items():
                Info = value
            Link = wl.text
print "Cache Name: ",CacheName
print "Cache ID: ",CacheID
print "Open Cache: ",OpenCache
print "Cache Type: ",CacheType
print "Cache Size: ",CacheSize
print "Difficulty: ", Difficulty
print "Terrain: ",Terrain
print "Lat: ",Lat
print "Lon: ",Lon
print "GType: ",GType
print "Link: ",Link
print "="*25

print "finished"
```

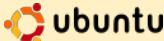
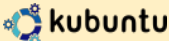





VEDI ANCHE:

FCM nn. 27-36 - Python parti 1-10

VALIDO PER:

CATEGORIE:

    
Sviluppo Grafica Internet M/media Sistema

DISPOSITIVI:

    
CD/DVD HDD USB Drive Laptop Wireless

L'ultima volta vi ho promesso che avremmo usato le nostre conoscenze XML per recuperare informazioni meteo da un sito web e mostrarle in un terminale. Bene, quel momento è arrivato.

Useremo una API da www.wunderground.com. Ho sentito la domanda "Cos'è una API" farsi strada nelle vostre gole. API sta per Interfaccia di Programmazione di

un'Applicazione. Si tratta di una espressione simpatica per descrivere il modo di interfacciarsi con un'altro programma. Pensate alle librerie che importiamo. Alcune di esse possono essere eseguite come applicazioni a sé stanti, ma se le importiamo come librerie possiamo usare molte delle loro funzioni nel nostro programma, riuscendo così ad usare il codice di qualcun altro. In questo caso, useremo indirizzi URL espressamente formattati per interrogare il sito wunderground per informazioni meteo - senza usare un browser web. Alcuni potrebbero affermare che una API è una specie di porta segreta per un altro programma - inserita intenzionalmente dal programmatore per farcela usare. Ad ogni modo, è una estensione supportata da un'applicazione per permetterne l'uso in altre applicazioni.

Suona intrigante? Bene, continuate a leggere, miei giovani padawan.

Avviate il vostro browser preferito e aprite il sito www.wunderground.com. Ora inserite il vostro codice postale o la città e lo stato (o paese) nel campo di ricerca. Ci sono molte informazioni qui. Ora, saltiamo alla pagina dell'API: http://wiki.wunderground.com/index.php/API_-_XML

Una delle prime cose che noterete sono le Condizioni di Utilizzo dell'API. Siete pregati di leggerle e attenervi ad esse. Non sono onerose e sono molto semplici da rispettare. Le cose che ci interessano sono le chiamate GeoLookupXML, WXCurrentObXML e ForecastXML. Prendetevi un po' di tempo per esaminarle.

Ho intenzione di evitare la routine GeoLookupXML, e lasciarvela vedere da soli. Ci concentreremo su due altri comandi: WXCurrentObXML (Condizioni Correnti) questa volta e ForecastXML (Previsione) la prossima.

Questo è il link per WXCurrentObXML: <http://api.wunderground.com/auto/wui/geo/WXCurrentObXML/index.xml?query=80013>

Sostituite il codice postale degli USA 80013 con il vostro o, se abitate al di fuori degli Stati Uniti, potete provare con città, paese - come Parigi, Francia, o Londra, Inghilterra.

E il link per ForecastXML: <http://api.wunderground.com/auto/wui/geo/ForecastXML/index.xml?query=80013>

Anche qui sostituite 80013 con il vostro codice postale o città, paese.

Iniziamo con le informazioni correnti. Incollate l'indirizzo nel vostro browser preferito. Vi saranno mostrate una gran quantità di informazioni. Lascero a voi decidere cosa è davvero importante, ma ci soffermeremo su alcuni elementi.

Per il nostro esempio, faremo

attenzione ai tag seguenti:

```
display_location
observation_time
weather
temperature_string
relative_humidity
wind_string
pressure_string
```

Naturalmente, potete aggiungere altri tag che per voi sono interessanti. Comunque, questi tag vi forniranno una panoramica abbastanza completa.

Ora che sappiamo cosa ci interessa, iniziamo a scrivere il codice della nostra applicazione. Osserviamo in generale il flusso complessivo del programma.

Prima di tutto, controlliamo cosa l'utente ci ha chiesto di fare. Se è stata fornita una località la useremo altrimenti utilizzeremo quella predefinita. Quindi passeremo alla funzione `getCurrents`. Usiamo la località per costruire la nostra stringa di richiesta da inviare al sito web. Useremo `urllib.urlopen` per recuperare la risposta dal web, metterla in un oggetto e passarlo alla libreria `ElementTree` per il parsing. Quindi chiuderemo la

connessione web e inizieremo a cercare i nostri tag. Una volta trovato un tag che ci interessa, salveremo questo testo in una variabile che possiamo usare in seguito per mostrare l'output. Quando avremo a disposizione tutti i dati li mostreremo. Abbastanza semplice in teoria.

Iniziamo col chiamare il nostro file `w_currents.py`. Ecco la parte di `import` del nostro codice:

```
from xml.etree import
ElementTree as ET

import urllib

import sys

import getopt
```

Quindi inseriremo una serie di linee di aiuto (in alto a destra) sopra gli `import`.

Assicuratevi di usare i tripli doppi apici. Ci permetterà di usare commenti multi-riga. Parleremo di questo tra poco.

Ora creeremo la nostra classe `stub`, in basso a destra, e le routine principali, che sono mostrate nella pagina seguente.

```
""" w_currents.py
Returns current conditions, forecast and alerts for a
given zipcode from WeatherUnderground.com.
Usage: python wonderground.py [options]
Options:
-h, --help Show this help
-l, --location City,State to use
-z, --zip Zipcode to use as location

Examples:
w_currents.py -h (shows this help information)
w_currents.py -z 80013 (uses the zip code 80013 as
location)
"""
```

```
class CurrentInfo:
"""
This routine retrieves the current condition xml data
from WeatherUnderground.com
based off of the zip code or Airport Code...
currently tested only with Zip Code and Airport code
For location,
if zip code use something like 80013 (no quotes)
if airport use something like "KDEN" (use double-quotes)
if city/state (US) use something like "Aurora,%20CO" or
"Aurora,CO" (use double-quotes)
if city/country, use something like "London,%20England"
(use double-quotes)
"""

def getCurrents(self,debuglevel,Location):
pass

def output(self):
pass

def DoIt(self,Location):
pass

=====
# END OF CLASS CurrentInfo()
=====
```

Ricorderete dagli articoli precedenti la riga "if __name__". Se la chiamiamo come un'applicazione a sé stante, la routine principale verrà eseguita, altrimenti la possiamo usare come parte di una libreria. Una volta nella routine principale, controlleremo, se è il caso, cosa è stato trasmesso.

Se l'utente usa il parametro "-h" o "-- help", mostreremo le triple righe d'aiuto all'inizio del codice del programma attraverso la chiamata a una routine di supporto che dice all'applicazione di mostrare __doc__.

Se l'utente usa "-l" (località) o "-z" (codice postale) l'informazione fornita sostituirà quella impostata internamente. Quando si passa una località, assicuratevi di usare i doppi apici per racchiudere la stringa e di non usare spazi. Per esempio, per recuperare le attuali condizioni di Dallas, Texas, usare -l "Dallas,Texas".

I lettori più astuti avranno già intuito che i controlli -z e -l sono pressoché la stessa cosa. Potete modificare -l per controllare la eventuale presenza di spazi e riformattare la stringa prima di

passarla alle routine. È qualcosa che ormai dovrete essere capaci di fare.

Per finire creiamo un'istanza della nostra classe CurrentInfo che chiamiamo currents, e quindi passiamo la località alla Routine "DoIt". Ecco come è fatta:

```
def DoIt(self,Location):  
  
self.getCurrents(1,Location)  
  
self.output()
```

Molto semplice. Passiamo la località e il livello di debug alla routine getCurrents, e quindi chiamiamo la routine di output. Anche se avremmo potuto semplicemente eseguire l'output direttamente dalla routine getCurrents, stiamo sviluppando la flessibilità per fornire l'output in differenti modi nel caso ne dovessimo aver bisogno.

Il codice per la routine getCurrents è mostrato nella pagina seguente.

Qui abbiamo un parametro chiamato debuglevel. Il suo utilizzo ci permette di controllare informazioni utili nel caso il

```
def usage():  
    print __doc__  
def main(argv):  
    location = 80013  
    try:  
        opts, args = getopt.getopt(argv, "hz:l:", ["help=",  
            "zip=", "location="])  
    except getopt.GetoptError:  
        usage()  
        sys.exit(2)  
    for opt, arg in opts:  
        if opt in ("-h", "--help"):  
            usage()  
            sys.exit()  
        elif opt in ("-l", "--location"):  
            location = arg  
        elif opt in ("-z", "--zip"):  
            location = arg  
    print "Location = %s" % location  
    currents = CurrentInfo()  
    currents.DoIt(location)  
  
    #=====  
    # Main loop  
    #=====  
if __name__ == "__main__":  
    main(sys.argv[1:])
```

programma non si comportasse nella maniera aspettata. È anche utile all'inizio della fase di sviluppo. Se poi siete contenti di come il vostro codice funziona, potete rimuovere ogni traccia relativa a debuglevel. Se avete intenzione di rilasciare il codice al pubblico, come se lo steste facendo per qualcun altro, assicuratevi di rimuovere il codice

e di testarlo nuovamente prima di rilasciarlo.

Adesso usiamo la funzione try/except per assicurarci che anche se qualcosa andasse storto, il programma non esploda. Nella sezione try impostiamo l'URL e un timer di 8 secondi (urllib.socket.setdefaulttimeout(8))

. Facciamo questo perché, a volte, wunderground è occupato e non risponde. In questa maniera si evita di dover aspettare all'infinito. Se siete interessati a ulteriori informazioni su urllib un buon posto da cui iniziare è <http://docs.python.org/library/urllib.html>.

Se accade qualcosa di inaspettato si passa nella sezione except che stampa un messaggio di errore e termina l'applicazione (sys.exit(2)).

Assumendo che tutto funzioni, iniziamo a cercare i nostri tag. La prima cosa che faremo è cercare la nostra località con tree.findall("//full"). Ricordate, tree è l'oggetto analizzato restituito da elementtree. In basso è mostrato parte di ciò che viene ritornato dall'API del sito.

Questa è la nostra prima istanza del tag <full>, che in questo caso è "Aurora, CO". Questa è la località che vogliamo usare. Quindi cerchiamo "observation_time". Rappresenta il momento di registrazione delle attuali condizioni. Continuiamo cercando tutti i dati che ci interessano - usando la stessa metodica.

Per finire ci occupiamo della nostra routine di output che è mostrata in alto a sinistra nella pagina seguente.

Qui semplicemente mostriamo le variabili.

Questo è tutto. Un output d'esempio per il mio codice postale con un debuglevel settato a 1 è mostrato in basso a sinistra nella prossima pagina.

```
<display_location>
<full>Aurora, CO</full>
<city>Aurora</city>
<state>CO</state>
<state_name>Colorado</state_name>
<country>US</country>
<country_iso3166>US</country_iso3166>
<zip>80013</zip>
<latitude>39.65906525</latitude>
<longitude>-104.78105927</longitude>
<elevation>1706.00000000 ft</elevation>
</display_location>
```

```
def getCurrents(self, debuglevel, Location):
    if debuglevel > 0:
        print "Location = %s" % Location
    try:
        CurrentConditions =
        'http://api.wunderground.com/auto/wui/geo/WXCurrentObXML
        /index.xml?query=%s' % Location
        urllib.socket.setdefaulttimeout(8)
        usock = urllib.urlopen(CurrentConditions)
        tree = ET.parse(usock)
        usock.close()
    except:
        print 'ERROR - Current Conditions - Could not get
        information from server...'
        if debuglevel > 0:
            print Location
            sys.exit(2)
        # Get Display Location
        for loc in tree.findall("//full"):
            self.location = loc.text
        # Get Observation time
        for tim in tree.findall("//observation_time"):
            self.obtime = tim.text
        # Get Current conditions
        for weather in tree.findall("//weather"):
            self.we = weather.text
        # Get Temp
        for TempF in tree.findall("//temperature_string"):
            self.tmpB = TempF.text
        #Get Humidity
        for hum in tree.findall("//relative_humidity"):
            self.relhum = hum.text
        # Get Wind info
        for windstring in tree.findall("//wind_string"):
            self.winds = windstring.text
        # Get Barometric Pressure
        for pressure in tree.findall("//pressure_string"):
            self.baroB = pressure.text
```

getCurrents routine

```
def output(self):
print 'Weather Information From Wunderground.com'
print 'Weather info for %s ' % self.location
print self.obtime
print 'Current Weather - %s' % self.we
print 'Current Temp - %s' % self.tmpB
print 'Barometric Pressure - %s' % self.baroB
print 'Relative Humidity - %s' % self.relhum
print 'Winds %s' % self.winds
```

Si prega di notare che ho scelto di usare i tag che includono i valori sia in Fahrenheit che Celsius. Se desiderate, per esempio, visualizzare solo i valori Celsius potete usare il tag `<temp_c>` piuttosto che `<temperature_string>`.

L'intero codice può essere scaricato da:
<http://pastebin.com/4ibJGm74>

```
Location = 80013
Weather Information From Wunderground.com
Weather info for Aurora, Colorado
Last Updated on May 3, 11:55 AM MDT
Current Weather - Partly Cloudy
Current Temp - 57 F (14 C)
Barometric Pressure - 29.92 in (1013 mb)
Relative Humidity - 25%
Winds From the WNW at 10 MPH
Script terminated.
```

La prossima volta ci concentreremo sulla parte previsionale delle API. Nel frattempo, buon divertimento!



Greg Walter 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.



Full Circle Podcast



Il **Podcast di Full Circle** è tornato e migliore che mai!

Gli argomenti nell'episodio quattro includono:

- News - rilasciato Ubuntu 10.04
 - Opinioni
 - Giochi - Steam arriva su Linux?
 - Feedback
- ...e tutto il solito umorismo.

I conduttori:

- * Robin Catling
- * Ed Hewitt
- * Dave Wilkins

Il podcast e le relative note li trovate su:
<http://fullcirclemagazine.org/>

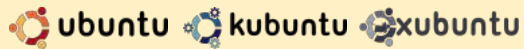




VEDI ANCHE:

FCM#27-37 - Python Parti 1 - 11

VALIDO PER:



CATEGORIE:



DISPOSITIVI:



web per le condizioni attuali, ce ne è uno per le previsioni. Ecco il link alla pagina delle previsioni XML:

<http://api.wunderground.com/auto/wui/geo/ForecastXML/index.xml?query=80013>

Come al solito, sostituite '80013' con la vostra Città/Paese, Città/Stato o codice postale. Probabilmente otterrete circa 600 righe di codice XML. C'è un elemento root chiamato 'forecast' e quindi quattro sotto elementi: 'termsofservice', 'txt_forecast', 'simpleforecast' and 'moon_phase'. Ci concentreremo su 'txt_forecast' e 'simpleforecast'.

Poiché l'ultima volta abbiamo già trattato delle sezioni usage, main e "if__name__", quelle le lascerò affrontare a voi e ci focalizzeremo sulle chicche di cui abbiamo bisogno questa volta. Poiché vi ho mostrato una porzione di txt_forecast, iniziamo da qui.

In basso è mostrata una piccola porzione di txt_forecast per la mia zona.

Dopo l'elemento precursore txt_forecast abbiamo la data, un elemento "number", quindi un elemento chiamato forecastday che ha a sua volta dei figli che includono period, icon, icons, title e qualcosa chiamato fcttext... e a seguire una ripetizione dello stesso. La prima cosa che noterete in txt_forecast è che la data non è una data ma un valore temporale. Si tratta del momento in cui è stata fatta la previsione. L'etichetta <number> mostra quante previsioni sono presenti

per le prossime 24 ore. Non ricordo di aver visto mai un valore inferiore a 2. Per ciascuna previsione nelle 24 ore (<forecastday>), avrete un <period> numerico, opzioni multiple per le icone, un'opzione per il titolo ("Today", "Tonight", "Tomorrow") ed il testo riassuntivo della previsione. Questa è un'anteprima rapida della previsione, di solito per le successive 12 ore.

Prima di iniziare a lavorare con il nostro codice dovremmo dare

```
<txt_forecast>
  <date>3:31 PM MDT</date>
  <number>2</number>
  -<forecastday>
    <period>1</period>
    <icon>nt_cloudy</icon>
    +<icons></icons>
    <title>Tonight</title>
    -<fcttext>
      Mostly cloudy with a 20
percent chance of thunderstorms in the evening...then
partly cloudy after midnight. Lows in the mid 40s.
Southeast winds 10 to 15 mph shifting to the south after
midnight.
    </fcttext>
  </forecastday>
  +<forecastday></forecastday>
</txt_forecast>
```

Nell'ultima sessione, abbiamo trattato l'API di wunderground e scritto un po' di codice per ricavare le condizioni attuali. Questa volta ci occuperemo della parte delle API riguardante le previsioni. Se non avete avuto modo di dare un'occhiata alle ultime due puntate su XML, ed in particolare l'ultima, dovrete farlo prima di procedere ulteriormente.

Così come esiste un indirizzo



un'occhiata alla porzione <simpleforecast> del file xml che è mostrato a destra.

C'è un'etichetta <forecastday> per ciascun giorno del periodo di previsione, di solito 6 giorni incluso quello attuale. Avete informazioni sulla data in vari formati (io personalmente preferisco l'etichetta <pretty>), temperature massime e minime previste sia in Fahrenheit che Celsius, proiezione delle condizioni generali, varie icone, una icona del cielo (condizioni del cielo alla stazione di riferimento), e "pop" che sta per "Probabilità di Precipitazione". L'etichetta <moon_phase> fornisce alcune informazioni interessanti come alba, tramonto e fasi lunari.

Ora ci addentriamo nel codice. Ecco la sezione degli import:

```
from xml.etree import  
ElementTree as ET
```

```
import urllib
```

```
import sys
```

```
import getopt
```

Dobbiamo ora creare la nostra classe. Creeremo una routine

`_init_` per impostare e pulire le variabili necessarie, come mostrato in alto a destra nella pagina seguente.

Se non vi interessa supportare sia Fahrenheit che Celsius, escludete la variabile che non vi interessa. Io ho deciso di tenerle entrambe.

A seguire, creeremo la nostra routine principale per recuperare i dati della previsione. È mostrata in basso a destra nella pagina successiva.

È molto simile alla routine della volta scorsa per le condizioni attuali. La differenza principale (finora) è l'URL utilizzato. Ora le cose cambiano. Dato che abbiamo figli multipli con la stessa etichetta all'interno del genitore dobbiamo modificare in parte le nostre chiamate all'analizzatore. Il codice è mostrato in alto a sinistra nella pagina seguente.

Notate come questa volta si stia usando `tree.find`, ed i cicli per scorrere i dati. È un peccato che Python non fornisca un costrutto SELECT/CASE come in altri linguaggi. La routine IF/ELIF,

```
<simpleforecast>  
  <-forecastday>  
    <period>1</period>  
    <-date>  
      <epoch>1275706825</epoch>  
      <pretty_short>9:00 PM MDT</pretty_short>  
      <pretty>9:00 PM MDT on June 04, 2010</pretty>  
      <day>4</day>  
      <month>6</month>  
      <year>2010</year>  
      <yday>154</yday>  
      <hour>21</hour>  
      <min>00</min>  
      <sec>25</sec>  
      <isdst>1</isdst>  
      <monthname>June</monthname>  
      <weekday_short/>  
      <weekday>Friday</weekday>  
      <amp;pm>PM</amp;pm>  
      <tz_short>MDT</tz_short>  
      <tz_long>America/Denver</tz_long>  
    </date>  
    <-high>  
      <fahrenheit>92</fahrenheit>  
      <celsius>33</celsius>  
    </high>  
    <-low>  
      <fahrenheit>58</fahrenheit>  
      <celsius>14</celsius>  
    </low>  
    <conditions>Partly Cloudy</conditions>  
    <icon>partlycloudy</icon>  
    +<icons>  
      <skyicon>partlycloudy</skyicon>  
      <pop>10</pop>  
    </forecastday>  
    ...  
</simpleforecast>
```

```

=====
# Get the forecast for today and (if available)
tonight
=====
fcst = tree.find('.//txt_forecast')
for f in fcst:
    if f.tag == 'number':
        self.periods = f.text
    elif f.tag == 'date':
        self.date = f.text
    for subelement in f:
        if subelement.tag == 'period':
            self.period=int(subelement.text)
        if subelement.tag == 'fcttext':

self.forecastText.append(subelement.text)
    elif subelement.tag == 'icon':
        self.icon.append( subelement.text)
    elif subelement.tag == 'title':
        self.Title.append(subelement.text)

```

```

class ForecastInfo:
    def __init__(self):
        self.forecastText = [] # Today/tonight forecast
information
        self.Title = [] # Today/tonight
        self.date = ''
        self.icon = [] # Icon to use for conditions
today/tonight
        self.periods = 0
        self.period = 0
=====
# Extended forecast information
=====
        self.extIcon = [] # Icon to use for extended
forecast
        self.extDay = [] # Day text for this forecast
("Monday", "Tuesday" etc)
        self.extHigh = [] # High Temp. (F)
        self.extHighC = [] # High Temp. (C)
        self.extLow = [] # Low Temp. (F)
        self.extLowC = [] # Low Temp. (C)
        self.extConditions = [] # Conditions text
        self.extPeriod = [] # Numerical Period information
(counter)
        self.extpop = [] # Percent chance Of
Precipitation

```

```

def GetForecastData(self,location):
    try:
        forecastdata = 'http://api.wunderground.com/auto/wui/geo/ForecastXML/index.xml?query=%s' % location
        urllib.socket.setdefaulttimeout(8)
        usock = urllib.urlopen(forecastdata)
        tree = ET.parse(usock)
        usock.close()
    except:
        print 'ERROR - Forecast - Could not get information from server...'
        sys.exit(2)

```

comunque, funziona a dovere, è solo un po' più arzigogolata. Ora analizziamo il codice. Assegnamo la variabile `fcst` ad ogni cosa all'interno dell'etichetta `<txt_forecast>` che così includerà tutti i dati per quel gruppo. Quindi cerchiamo le etichette `<date>` e `<number>` — dato che sono semplici etichette di "primo livello" — e carichiamo i dati nelle nostre variabili. Ora le cose si fanno un po' più difficili. Guardate al nostro esempio xml di risposta. Ci sono due istanze di `<forecastday>`. Sotto `<forecastday>` ci sono i sotto-elementi `<period>`, `<icon>`, `<icons>`, `<title>` e `<fcttext>`. Li passeremo in rassegna ed ancora utilizzeremo l'istruzione IF per caricarli nelle nostre variabili.

Ora dobbiamo dare un'occhiata ai dati sulle previsioni estese per i successivi X giorni. Fondamentalmente ricorriamo allo stesso metodo usato per assegnare le nostre variabili; questo è mostrato in alto a destra.

Adesso dobbiamo creare la nostra routine per l'output. Come abbiamo fatto l'ultima volta, sarà abbastanza generico. Il codice per fare ciò è mostrato a destra nella

pagina seguente.

Ancora, se non volete mostrare le informazioni sia in Centigradi che in Fahrenheit, modificate il codice per mostrare quello che volete. Per finire abbiamo la routine "DoIt":

```
def  
DoIt(self, Location, US, IncludeTo  
day, Output):
```

```
self.GetForecastData(Location)
```

```
self.output(US, IncludeToday, Out  
put)
```

Ora possiamo chiamare la routine come segue:

```
forecast = ForecastInfo()
```

```
forecast.DoIt('80013', 1, 0, 0) #  
Insert your own postal code
```

Questo è tutto per ora. Lascio il codice per i controlli a voi, se volete cimentarvi.

Ecco il codice eseguibile completo:

<http://pastebin.com/wsSXMxQx>

Divertitevi fino alla prossima volta.

```
#####  
# Now get the extended forecast  
#####  
fcst = tree.find('./simpleforecast')  
for f in fcst:  
    for subelement in f:  
        if subelement.tag == 'period':  
            self.extPeriod.append(subelement.text)  
        elif subelement.tag == 'conditions':  
            self.extConditions.append(subelement.text)  
        elif subelement.tag == 'icon':  
            self.extIcon.append(subelement.text)  
        elif subelement.tag == 'pop':  
            self.extpop.append(subelement.text)  
        elif subelement.tag == 'date':  
            for child in subelement.getchildren():  
                if child.tag == 'weekday':  
                    self.extDay.append(child.text)  
        elif subelement.tag == 'high':  
            for child in subelement.getchildren():  
                if child.tag == 'fahrenheit':  
                    self.extHigh.append(child.text)  
                if child.tag == 'celsius':  
                    self.extHighC.append(child.text)  
        elif subelement.tag == 'low':  
            for child in subelement.getchildren():  
                if child.tag == 'fahrenheit':  
                    self.extLow.append(child.text)  
                if child.tag == 'celsius':  
                    self.extLowC.append(child.text)
```



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.

```

def output(self,US,IncludeToday,Output):
    # US takes 0,1 or 2
    # 0 = Centigrade
    # 1 = Fahrenheit
    # 2 = both (if available)
    # Now print it all
    if Output == 0:
        for c in range(int(self.period)):
            if c <> 1:
                print '-----'
                print 'Forecast for %s' %
self.Title[c].lower()
                print 'Forecast = %s' %
self.forecastText[c]
                print 'ICON=%s' % self.icon[c]
                print '-----'
            print 'Extended Forecast...'
            if IncludeToday == 1:
                startRange = 0
            else:
                startRange = 1
            for c in range(startRange,6):
                print self.extDay[c]
                if US == 0: #Centigrade information
                    print '\tHigh - %s(C)' %
self.extHigh[c]
                    print '\tLow - %s(C)' % self.extLow[c]
                elif US == 1: #Fahrenheit information
                    print '\tHigh - %s(F)' %
self.extHigh[c]
                    print '\tLow - %s(F)' % self.extLow[c]
                else: #US == 2 both(if available)
                    print '\tHigh - %s' % self.extHigh[c]
                    print '\tLow - %s' % self.extLow[c]
                if int(self.extpop[c]) == 0:
                    print '\tConditions - %s.' %
self.extConditions[c]
                else:
                    print '\tConditions - %s. %d%% chance
of precipitation.' %
(self.extConditions[c],int(self.extpop[c]))

```



Full Circle Podcast



Il **Podcast di Full Circle** è tornato e migliore che mai!

Gli argomenti nell'episodio otto includono:

- News - sviluppo di Maverick
- Intervista a Lubuntu
- Giochi - Ed recensisce Osmos
- Feedback

...e tutto il solito umorismo.

I conduttori:

- *Robin Catling*
- *Ed Hewitt*
- *Dave Wilkins*

Il podcast e le relative note li trovate su:

<http://fullcirclemagazine.org/>



Questo mese parleremo dell'uso di Curses in Python. No, non stiamo dicendo di usare Python per dire parolacce, benché possiate in caso ne sentiate il bisogno. Stiamo parlando dell'utilizzo della libreria Curses per creare qualche stravagante output dello schermo.

Se siete vecchi abbastanza da ricordare i primi tempi dei computer, ricorderete che, in ambito lavorativo, i computer erano tutti mainframe - con semplici terminali (schermi e tastiere) per l'input e l'output. Potevate avere più terminali collegati ad un solo computer. Il problema era che i terminali risultavano dispositivi noiosi. Non avevano finestre, colori o cose del genere - solo 24 righe di 80 caratteri (nelle migliori delle ipotesi). Quando i personal computer divennero popolari, ai tempi di DOS e CPM, questo è quello che ci si ritrovava. Quando i programmatori lavoravano sui loro stravaganti (per quei tempi) schermi, soprattutto per

l'immissione e visualizzazione di dati, usavano carta grafica per disegnare il contenuto dello schermo. Ciascun blocco sulla carta grafica rappresentava la posizione di un carattere. Quando usiamo i nostri programmi in Python nel terminale, ancora abbiamo a che fare con uno schermo 24x80. Comunque, quella limitazione può essere facilmente superata con un'adeguata accortezza e preparazione. Quindi recatevi verso il più vicino negozio di articoli di cancelleria e procuratevi qualche blocchetto di carta grafica.

Ad ogni modo, prepariamoci a creare il nostro primo programma Curses, mostrato in alto a destra. Ve lo spiegherò dopo che avrete dato un'occhiata al codice.

Breve ma semplice. Esaminiamolo riga per riga. Per iniziare, inseriamo gli import, che ormai dovrebbero esservi familiari. In seguito, creeremo un nuovo oggetto Curser per lo schermo, lo inizializzeremo e lo chiameremo `myscreen`. (`myscreen =`

```
#!/usr/bin/env python
# CursesExample1
#-----
# Curses Programming Sample 1
#-----
import curses
myscreen = curses.initscr()
myscreen.border(0)
myscreen.addstr(12, 25, "See Curses, See Curses Run!")
myscreen.refresh()
myscreen.getch()
curses.endwin()
```

`curses.initscr()`). Questa è la nostra tela che dipingeremo. Quindi, usiamo il comando `myscreen.border(0)` per disegnare un margine intorno alla tela. Non è necessario ma rende lo schermo più gradevole. Quindi usiamo il metodo `addstr` per "scrivere" sulla nostra tela a partire dalla riga 12 posizione 25. Pensate al metodo `.addstr` di Curses come ad un'istruzione `print`. Per finire, il metodo `.refresh()` renderà il nostro lavoro visibile. Se non ricarichiamo lo schermo, i nostri cambiamenti non saranno visibili. Quindi aspettiamo che l'utente preme un tasto (`.getch`) e poi rilasciamo l'oggetto schermo (`.endwin`) per permettere al nostro terminale di

tornare alla normalità. Il comando `curses.endwin()` è MOLTO importante e, se non viene richiamato, il vostro terminale risulterà in uno stato di grande disordine. Assicuratevi quindi di usare questo metodo prima della fine dell'applicazione.

Salvate questo programma come `CursesExample1.py` ed eseguitelo in un terminale. Alcune cose da notare. Ogniqualvolta usate un bordo, questo spreca un carattere "utilizzabile" per ciascun elemento del bordo stesso. In aggiunta, sia la posizione della linea che del carattere viene conteggiata a partire da ZERO.

Questo significa che la prima riga sul nostro schermo è la linea 0 e l'ultima è la 23. Così, la posizione in alto a sinistra è riferita 0,0 e quella in basso a destra è 23,79. Facciamo un piccolo esempio (in alto a destra) per dimostrare questo.

Tutto molto semplice eccetto che per i blocchi try/finally. Ricordate, ho detto che `curses.endwin` è MOLTO importante e deve essere richiamato prima che l'applicazione termini. Bene, in questa maniera, anche se le cose si mettono male, la routine `endwin` verrà richiamata. Ci sono molti modi di farlo, ma questo mi sembra abbastanza semplice.

Ora creiamo un menu di sistema carino. Se ricordate tempo addietro realizzammo

l'applicazione `cookbook` che aveva un menu (Programmare in Python - Parte 8). Tutto nel terminale scorreva semplicemente verso l'alto quando stampavamo a schermo qualcosa. Questa volta prenderemo quell'idea e creeremo un semplice menu che potrete utilizzare per abbellire `cookbook`. In basso è mostrato ciò che utilizzammo allora.

Questa volta useremo `Curses`. Iniziamo con il seguente modello. Probabilmente vorrete salvare questo frammento di codice (in basso a destra) per poterlo utilizzare nei vostri programmi futuri.

Ora salvate ancora il vostro

```
=====
                        RECIPE DATABASE
=====
1 - Show All Recipes
2 - Search for a recipe
3 - Show a Recipe
4 - Delete a recipe
5 - Add a recipe
6 - Print a recipe
0 - Exit
=====
Enter a selection ->
```

```
#!/usr/bin/env python
# CursesExample2
import curses
=====
#                               MAIN LOOP
=====
try:
    myscreen = curses.initscr()
    myscreen.clear()
    myscreen.addstr(0,0,"0          1          2          3
4          5          6          7")
    myscreen.addstr(1,0,"123456789012345678901234567890123456789012345678901
234567890123456789012345678901234567890")
    myscreen.addstr(10,0,"10")
    myscreen.addstr(20,0,"20")
    myscreen.addstr(23,0, "23 - Press Any Key to Continue")
    myscreen.refresh()
    myscreen.getch()
finally:
    curses.endwin()
```

```
#!/usr/bin/env python
#-----
# Curses Programming Template
#-----
import curses

def InitScreen(Border):
    if Border == 1:
        myscreen.border(0)

#-----
#                               MAIN LOOP
=====
myscreen = curses.initscr()
InitScreen(1)
try:
    myscreen.refresh()
    # Your Code Stuff Here...
    myscreen.addstr(1,1, "Press Any Key to Continue")
    myscreen.getch()
finally:
    curses.endwin()
```

modello come "cursesmenu1.py" cosicché possiamo lavorare sul file e mantenere il modello.

Prima di procedere ulteriormente con il codice, prepariamoci ad un approccio modulare. Qui (in alto a destra) c'è un esempio in pseudo-codice di quello che andremo a realizzare.

Ovviamente questo pseudo-codice è solo... pseudo. Ma vi dà un'idea sull'indirizzo del progetto generale. Dato che si tratta solo di un esempio, non ci dilungheremo oltre in questa sede, ma siete liberi di portarlo a compimento. Iniziamo con il ciclo principale (al centro, all'estrema destra).

Non c'è molto da programmare qui. Abbiamo i nostri blocchi try|finally così come li avevamo nel modello. Inizializziamo lo schermo Curses e quindi richiamiamo una routine chiamata LogicLoop. Il suo codice è mostrato in basso, all'estrema destra.

Ancora, non molto, ma è solo un esempio. Qui invocheremo due funzioni. Una chiamata DoMainMenu e l'altra MainInKey. DoMainMenu mostrerà il nostro

```
curses.initscr()
LogicLoop
    ShowMainMenu          # Show the main menu
    MainInKey             # This is our main input handling routine
        While Key != 0:
            If Key == 1:
                ShowAllRecipesMenu # Show the All Recipes Menu
                Inkey1             # Do the input routines for this
                ShowMainMenu       # Show the main menu
            If Key == 2:
                SearchForARecipeMenu # Show the Search for a Recipe Menu
                InKey2              # Do the input routines for this option
                ShowMainMenu       # Show the main menu again
            If Key == 3:
                ShowARecipeMenu    # Show the Show a recipe menu routine
                InKey3             # Do the input routine for this routine
                ShowMainMenu       # Show the main menu again
            ...                  # And so on and so on
curses.endwin()          # Restore the terminal
```

```
def DoMainMenu():
    myscreen.erase()
    myscreen.addstr(1,1,
"=====")
    myscreen.addstr(2,1, "          Recipe
Database")
    myscreen.addstr(3,1,
"=====")
    myscreen.addstr(4,1, "  1 - Show All
Recipes")
    myscreen.addstr(5,1, "  2 - Search for a
recipe")
    myscreen.addstr(6,1, "  3 - Show a recipe")
    myscreen.addstr(7,1, "  4 - Delete a recipe")
    myscreen.addstr(8,1, "  5 - Add a recipe")
    myscreen.addstr(9,1, "  6 - Print a recipe")
    myscreen.addstr(10,1, "  0 - Exit")
    myscreen.addstr(11,1,
"=====")
    myscreen.addstr(12,1, "  Enter a selection: ")
    myscreen.refresh()
```

```
#    MAIN LOOP
try:
    myscreen = curses.initscr()
    LogicLoop()
finally:
    curses.endwin()
```

```
def LogicLoop():
    DoMainMenu()
    MainInKey()
```


menu principale, mentre MainInKey gestirà ogni altra cosa di questo menu principale. La routine DoMainMenu è mostrata a destra.

Notate come questa funzione non faccia altro che pulire lo schermo (myscreen.erase) e quindi stampare quello che vogliamo. Non c'è niente qui che si occupi della gestione della tastiera. Questo è il lavoro della routine MainInKey, che è mostrata in basso.

Si tratta di una funzione davvero semplice. Si entra in un ciclo while finché il tasto premuto dall'utente è uguale a 0. All'interno del ciclo, controlliamo se è uguale a diversi valori e, se così, lanciamo una serie di funzioni e quando finito richiamiamo il menu principale. Potete completare la maggior parte di queste funzioni da soli, ma daremo un'occhiata all'opzione 2, Search for a Recipe. Questo menu è breve e grazioso.

```
def MainInKey():
    key = 'X'
    while key != ord('0'):
        key = myscreen.getch(12,22)
        myscreen.addch(12,22,key)
        if key == ord('1'):
            ShowAllRecipesMenu()
            DoMainMenu()
        elif key == ord('2'):
            SearchForARecipeMenu()
            InKey2()
            DoMainMenu()
        elif key == ord('3'):
            ShowARecipeMenu()
            DoMainMenu()
        elif key == ord('4'):
            NotReady("'Delete A Recipe'")
            DoMainMenu()
        elif key == ord('5'):
            NotReady("'Add A Recipe'")
            DoMainMenu()
        elif key == ord('6'):
            NotReady("'Print A Recipe'")
            DoMainMenu()
    myscreen.refresh()
```

```
def SearchForARecipeMenu():
    myscreen.addstr(4,1, "-----")
    myscreen.addstr(5,1, " Search in")
    myscreen.addstr(6,1, "-----")
    myscreen.addstr(7,1, " 1 - Recipe Name")
    myscreen.addstr(8,1, " 2 - Recipe Source")
    myscreen.addstr(9,1, " 3 - Ingredients")
    myscreen.addstr(10,1, " 0 - Exit")
    myscreen.addstr(11,1, "Enter Search Type -> ")
    myscreen.refresh()

def InKey2():
    key = 'X'
    doloop = 1
    while doloop == 1:
        key = myscreen.getch(11,22)
        myscreen.addch(11,22,key)
        tmpstr = "Enter text to search in "
        if key == ord('1'):
            sstr = "'Recipe Name' for -> "
            tmpstr = tmpstr + sstr
            retstring = GetSearchLine(13,1,tmpstr)
            break
        elif key == ord('2'):
            sstr = "'Recipe Source' for -> "
            tmpstr = tmpstr + sstr
            retstring = GetSearchLine(13,1,tmpstr)
            break
        elif key == ord('3'):
            sstr = "'Ingredients' for -> "
            tmpstr = tmpstr + sstr
            retstring = GetSearchLine(13,1,tmpstr)
            break
        else:
            retstring = ""
            break
    if retstring != "":
        myscreen.addstr(15,1, "You entered - " + retstring)
    else:
        myscreen.addstr(15,1, "You entered a blank string")
    myscreen.refresh()
    myscreen.addstr(20,1, "Press a key")
    myscreen.getch()

def GetSearchLine(row,col,strng):
    myscreen.addstr(row,col,strng)
    myscreen.refresh()
    instrng = myscreen.getstr(row,len(strng)+1)
    myscreen.addstr(row,len(strng)+1,instrng)
    myscreen.refresh()
    return instrng
```

La funzione `InKey2` (a destra) è un po' più complicata.

Ancora, stiamo usando un ciclo `while` standard. Impostiamo la variabile `doloop = 1`, cosicché il nostro ciclo continui senza fine finché non troviamo quello che vogliamo. Usiamo il comando `break` per interrompere il ciclo. Le tre opzioni sono molto simili. La differenza sostanziale è che iniziamo con una variabile chiamata `tmpstr`, e poi aggiungiamo una qualunque opzione che abbiamo scelto... rendendola un po' più amichevole. Quindi chiamiamo la funzione `GetSearchLine` per ottenere la stringa da cercare. Usiamo la routine `getstr` per ricevere dall'utente una stringa piuttosto che un carattere. Quindi restituiamo la stringa alla funzione di input per l'elaborazione successiva.

Il codice completo è all'indirizzo:

<http://pastebin.com/ELuZ3T4P>

Un'ultima cosa. Se siete interessati ad approfondire Curses, ci sono molti altri metodi disponibili oltre a quelli utilizzati

questo mese. Oltre a fare una ricerca con Google, il miglior punto di partenza è la documentazione ufficiale:

<http://docs.python.org/library/curses.html>.

Alla prossima.

OOPS!

Sembra che il codice per **Python Pt.11** non sia stato stampato correttamente su Pastebin. L'indirizzo corretto per il codice è:
<http://pastebin.com/Pk74fLF3>

Consultate anche:
<http://fullcirclemagazine.pastebin.com> per tutto il codice Python (anche futuro).



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.



Full Circle Podcast



AUDIO MP3



AUDIO OGG

Il **Podcast di Full Circle** è tornato e migliore che mai!

Gli argomenti nell'episodio dieci includono:

- News
- Opinione - Contribuire agli articoli con il redattore di FCM.
- Intervista con Amber Graner
- Feedback
- ...e tutto il solito umorismo.

I conduttori:

- *Robin Catling*
- *Ed Hewitt*
- *Ronnie Tucker*

Il podcast e le relative note li trovate su:

<http://fullcirclemagazine.org/>





L'ultima volta abbiamo trattato la libreria Curses. Questa volta approfondiremo l'argomento concentrandoci sui comandi per il colore. Se vi siete persi l'articolo precedente, ecco un breve riassunto. Prima di tutto dovete importare la libreria curses. Quindi dovete inizializzare con `curses.initscr()`. Per inserire il testo sullo schermo si chiama la funzione `addstr` e quindi `refresh` per mostrare i cambiamenti. Infine chiamate `curses.endwin()` per ripristinare la finestra del terminale al suo stato normale.

Ora creeremo un programma semplice e veloce che usa i colori. È molto simile a quanto già fatto ma con l'aggiunta di nuovi comandi. Si usa prima `curses.start_color()` per dire al sistema che vogliamo usare i colori nel nostro programma. Quindi assegniamo la coppia di colori per il testo e lo sfondo. Possiamo assegnare più coppie e scegliere quale usare di volta in volta. L'assegnazione avviene tramite la funzione `curses.init_pair()`. La sintassi è:

```
curses.init_pair([pairnumber]
,[foreground
color],[background color])
```

I colori sono impostati usando "curses.COLOR_" con il colore desiderato. Per esempio, `curses.COLOR_BLUE` o `curses.COLOR_GREEN`. I possibili valori sono black, red, green, yellow, blue, magenta, cyan e white. Basta aggiungere in maiuscolo il nome del colore a "curses.COLOR_". Una volta definite le nostre coppie di colori, possiamo usarle come parametro finale della funzione `screen.addstr` come nella seguente riga:

```
myscreen.addstr([row],[column]
],[text],curses.color_pair(X)
)
```

Qui X rappresenta il set di colori che vogliamo usare.

Salvate il seguente codice (in alto a destra) come `colortest1.py` quindi eseguitelo. Non cercate di eseguire un programma curses in una IDE come SPE o Dr. Python. Eseguite in un terminale.

Ciò che dovrete vedere è uno

```
import curses
try:
    myscreen = curses.initscr()
    curses.start_color()
    curses.init_pair(1, curses.COLOR_BLACK,
curses.COLOR_GREEN)
    curses.init_pair(2, curses.COLOR_BLUE,
curses.COLOR_WHITE)
    curses.init_pair(3,
curses.COLOR_MAGENTA,curses.COLOR_BLACK)
    myscreen.clear()
    myscreen.addstr(3,1," This is a test
",curses.color_pair(1))
    myscreen.addstr(4,1," This is a test
",curses.color_pair(2))
    myscreen.addstr(5,1," This is a test
",curses.color_pair(3))
    myscreen.refresh()
    myscreen.getch()
finally:
    curses.endwin()
```

sfondo grigio con tre righe di testo recanti la scritta "This is a test" in colori differenti. Il primo dovrebbe essere nero-su-verde, il secondo blu-su-bianco e il terzo magenta sullo sfondo grigio.

Ricordate la coppia Try/Finally. Il suo utilizzo ci permette di ripristinare il terminale al suo stato normale se qualcosa di sbagliato si dovesse verificare. In alternativa è possibile usare il comando curses

chiamato wrapper. Wrapper esegue il lavoro al posto vostro. Eseguite `curses.initscr()`, `curses.start_color()` e `curses.endwin()` così da non doverlo fare voi stessi. Vi dovete solo ricordare di richiamare `curses.wrapper` nella vostra funzione main. Questa restituisce un puntatore allo schermo. Nella pagina seguente (in alto a destra) c'è lo stesso programma appena creato ma con la funzione `curses.wrapper`.

Come vedete è molto più semplice e non abbiamo bisogno di chiamare `curses.endwin()` se si verifica qualche errore. Viene fatto tutto al posto nostro.

Ora che abbiamo una infarinatura delle basi, mettiamo a frutto quello che abbiamo imparato nell'ultimo anno di lavoro e creiamo un gioco. Prima di iniziare, definiamo quello che andremo a fare. Il nostro gioco prenderà una lettera maiuscola qualunque e la muoverà dal lato destro al sinistro dello schermo. A una posizione casuale questa cambierà direzione dirigendosi verso il basso. Noi avremo un "cannone" che potrà essere mosso tramite i tasti freccia destra e sinistra così da posizionarlo sotto la lettera in caduta. Quindi premendo la barra spazio potremo sparare. Se colpiremo la lettera prima che questa raggiunga il nostro cannone realizzeremo un punto. Altrimenti il nostro cannone esploderà. Se perdiamo tre cannoni il gioco finirà. Anche se può sembrare un gioco semplice in realtà c'è parecchio codice da scrivere.

Iniziamo. Dobbiamo preparare il nostro lavoro e creare alcune

funzioni prima di procedere ulteriormente. Creiamo un nuovo progetto e chiamiamolo `game1.py`. Iniziamo con il codice mostrato in basso a destra:

Questo codice al momento non fa granché, ma è il nostro punto di inizio. Notate che abbiamo quattro istruzioni `init_pair` per definire i colori che useremo come nostri set casuali e uno per le esplosioni (numero 5). Ora dobbiamo impostare alcune variabili e costanti che useremo durante il gioco. Le inseriremo nella routine `__init__` della classe `Game1`. Sostituite l'istruzione `pass` in `__init__` con il codice della pagina seguente.

Dovreste essere in grado di capire cosa accade in queste definizioni. Se al momento dovessero sorgere dubbi, tutto dovrebbe diventare chiaro più avanti.

Siamo vicini ad ottenere qualcosa di eseguibile. Mancano ancora poche routine. Lavoriamo sulla funzione che muove una lettera da destra a sinistra sullo schermo:

<http://fullcirclemagazine.pastebin.com/z5CgMAGm>

```
import curses
def main(stdscreen):
    curses.init_pair(1, curses.COLOR_BLACK,
curses.COLOR_GREEN)
    curses.init_pair(2, curses.COLOR_BLUE,
curses.COLOR_WHITE)
    curses.init_pair(3,
curses.COLOR_MAGENTA, curses.COLOR_BLACK)
    stdscreen.clear()
    stdscreen.addstr(3,1, " This is a test
", curses.color_pair(1))
    stdscreen.addstr(4,1, " This is a test
", curses.color_pair(2))
    stdscreen.addstr(5,1, " This is a test
", curses.color_pair(3))
    stdscreen.refresh()
    stdscreen.getch()
curses.wrapper(main)
```

```
import curses
import random

class Game1():
    def __init__(self):
        pass
    def main(self, stdscr):
        curses.init_pair(1, curses.COLOR_BLACK,
curses.COLOR_GREEN)
        curses.init_pair(2, curses.COLOR_BLUE,
curses.COLOR_BLACK)
        curses.init_pair(3, curses.COLOR_YELLOW,
curses.COLOR_BLUE)
        curses.init_pair(4, curses.COLOR_GREEN,
curses.COLOR_BLUE)
        curses.init_pair(5, curses.COLOR_BLACK,
curses.COLOR_RED)

    def StartUp(self):
        curses.wrapper(self.main)
g = Game1()
g.StartUp()
```

Questa è la routine più corposa dell'intero programma, e a sua volta introduce nuove funzioni. `scrn.delch()` cancella il carattere a una data riga/colonna. `curses.napms()` dice a python di fermarsi per X millisecondi (ms).

La logica della routine si trova (in pseudocodice) nella pagina seguente (in alto a destra).

Ora dovrete essere in grado di seguire il codice. Abbiamo bisogno di due nuove routine per garantire che tutto sia corretto. La prima è `Explode`, in cui inseriremo la direttiva `pass`. La seconda è `ResetForNew`. Qui ripristineremo la riga corrente per la lettera alla riga predefinita, la colonna corrente, imposteremo `DroppingLetter` a 0, prenderemo una lettera e un punto di caduta casuali. Nella pagina che segue, al centro a destra sono presentate le due funzioni.

Abbiamo ora bisogno di altre quattro funzioni per mantenere il tutto (pagina seguente, in basso a destra). Una per la lettera casuale, un'altra per il punto di caduta. Ricordate che abbiamo discusso velocemente il modulo `random` all'inizio della serie.

In `PickALetter` generiamo un intero casuale tra 65 e 90 (da "A" a "Z"). Se ricordate, quando si usa la funzione `random` bisogna fornire un valore minimo e uno massimo. Stessa cosa per `PickDropPoint`. Eseguiamo anche una chiamata a `random.seed()` in entrambe le

routine che assegna al generatore casuale un numero diverso ad ogni chiamata. La quarta funzione è `CheckKeys`. Questa si occupa di controllare i tasti premuti dall'utente e tradurli in movimenti del cannone. Comunque per il momento non la completeremo, lo faremo più tardi. Abbiamo bisogno

anche della funzione `CheckForHit`, anche questa solo dichiarata.

```
def
CheckKeys(self,scrn,keyin):
    pass
def CheckForHit(self,scrn):
    pass
```

Ora creeremo una breve

```
# Line Specific Stuff
self.GunLine = 22                #Row where our gun lives
self.GunPosition = 39           #Where the gun starts on GunLine
self.LetterLine = 2             #Where our letter runs right to left
self.ScoreLine = 1              #Where we are going to display the score
self.ScorePosition = 50         #Where the score column is
self.LivesPosition = 65         #Where the lives column is

# Letter Specific Stuff
self.CurrentLetter = "A"        #A dummy Holder Variable
self.CurrentLetterPosition = 78 #Where the letter will start on the LetterLine
self.DropPosition = 10          #A dummy Holder Variable
self.DroppingLetter = 0         #Flag - Is the letter dropping?
self.CurrentLetterLine = 3      #A dummy Holder Variable
self.LetterWaitCount = 15       #How many times should we loop before actually
                                #working?

# Bullet Specific Stuff
self.Shooting = 0               #Flag - Is the gun shooting?
self.BulletRow = self.GunLine - 1
self.BulletColumn = self.GunPosition

# Other Stuff
self.LoopCount = 0              #How many loops have we done in MoveLetter
self.GameScore = 0              #Current Game Score
self.Lives = 3                  #Default number of lives
self.CurrentColor = 1           #A dummy Holder Variable
self.DecScoreOnMiss = 0         #Set to 1 if you want to decrement the
                                #score every time the letter hits the
                                #bottom row
```

funzione che sarà il "cervello" del nostro gioco. La chiameremo GameLoop (pagina seguente, in alto a destra).

La logica che sta dietro si basa nell'impostare la nostra tastiera su nodelay(1). Questo significa che non attenderemo che un tasto venga premuto e quando accade lo registriamo per un utilizzo successivo. Quindi entriamo in un ciclo while forzato a essere sempre vero (1) cosicché il gioco vada avanti finché non siamo pronti a smettere. Mettiamo in pausa per 40 millisecondi quindi muoviamo la nostra lettera e controlliamo se l'utente ha premuto un tasto. Se è una "Q" (notate che è in maiuscolo) o il tasto ESC allora interrompiamo il ciclo e terminiamo il programma. Altrimenti controlliamo se si tratta del tasto freccia destra o sinistra o la barra spazio. Successivamente è possibile rendere il gioco più complesso confrontando il tasto premuto con il carattere corrente e quindi sparare solo se l'utente ha premuto lo stesso tasto, come un programma per migliorare la dattilografia. Ricordatevi di rimuovere "Q" come tasto di chiusura.

Abbiamo anche bisogno di una

```
IF we have waited the correct number of loops THEN
  Reset the loop counter
  IF we are moving to the left of the screen THEN
    Delete the character at the the current row,column.
    Sleep for 50 milliseconds
    IF the current column is greater than 2 THEN
      Decrement the current column
    Set the character at the current row,column
    IF the current column is at the random column to drop to the bottom THEN
      Set the DroppingLetter flag to 1
  ELSE
    Delete the character at the current row,column
    Sleep for 50 milliseconds
    IF the current row is less than the line the gun is on THEN
      Increment the current row
      Set the character at the current row,column
    ELSE
      IF
        Explode (which includes decrementing the score if you wish) and check to
        see if we continue.
        Pick a new letter and position and start everything over again.
  ELSE
    Increment the loopcounter
    Refresh the screen.
```

funzione che imposti ciascuna nuova partita. La chiamiamo NewGame (pagina seguente, al centro a destra).

Necessitiamo anche della routine PrintScore che mostra il punteggio corrente e il numero di vite restanti (pagina seguente, in basso a destra).

Ora dobbiamo solo aggiungere un po' di codice (pagina seguente, in basso a sinistra) alla nostra funzione

```
def Explode(self,scrn):
    pass
def ResetForNew(self):
    self.CurrentLetterLine = self.LetterLine
    self.CurrentLetterPosition = 78
    self.DroppingLetter = 0
    self.PickALetter()
    self.PickDropPoint()
```

```
def PickALetter(self):
    random.seed()
    char = random.randint(65,90)
    self.CurrentLetter = chr(char)
```

```
def PickDropPoint(self):
    random.seed()
    self.DropPosition = random.randint(3,78)
```

PROGRAMMARE IN PYTHON - PARTE 14

main per iniziare il ciclo. Altro codice è in basso. Aggiungetelo sotto l'ultima chiamata a `init_pair`.

Ora dovremmo avere un programma che fa qualcosa. Provatelo. Io aspetto.

Il programma prende una lettera maiuscola casuale, la muove da destra a sinistra dello schermo per un numero casuale di colonne quindi la muove verso il basso. Comunque una cosa che noterete e che ad ogni avvio del programma la prima lettera è sempre una "A" e il punto di caduta è sempre la colonna 10. Questo per via dei valori predefiniti in `__init__`. Per

risolvere chiamate `self.ResetForNew` prima di entrare nel ciclo while nella funzione `Main`.

A questo punto dobbiamo lavorare sul nostro "cannone" e le routine di supporto. Aggiungete il codice (pagina seguente, in alto a destra) alla classe `Game1`.

`Movegun` si occupa di muovere il cannone in qualunque direzione si voglia. L'unica sezione nuova in questa routine è la funzione `addch`. Chiamiamo `colorpair(2)` per definire il colore e, allo stesso tempo, forziamo il cannone ad avere l'attributo `grassetto`. Usiamo l'operatore di bitwise OR (`|`) per

```
stdscr.addstr(11,28,"Welcome to Letter Attack")
stdscr.addstr(13,28,"Press a key to begin...")
stdscr.getch()
stdscr.clear()
PlayLoop = 1
while PlayLoop == 1:
    self.NewGame(stdscr)
    self.GameLoop(stdscr)
    stdscr.nodelay(0)
    curses.flushinp()
    stdscr.addstr(12,35,"Game Over")
    stdscr.addstr(14,23,"Do you want to play
again? (Y/N)")
    keyin = stdscr.getch(14,56)
    if keyin == ord("N") or keyin == ord("n"):
        break
    else:
        stdscr.clear()
```

```
def GameLoop(self,scrn):
    test = 1 #Set the loop
    while test == 1:
        curses.napms(20)
        self.MoveLetter(scrn)
        keyin =
scrn.getch(self.ScoreLine,self.ScorePosition)
        if keyin == ord('Q') or keyin == 27: # 'Q'
or <Esc>
            break
        else:
            self.CheckKeys(scrn,keyin)
            self.PrintScore(scrn)
            if self.Lives == 0:
                break
    curses.flushinp()
    scrn.clear()
```

```
def NewGame(self,scrn):
    self.GunChar = curses.ACS_SSBS

scrn.addch(self.GunLine,self.GunPosition,self.GunChar,cu
rses.color_pair(2) | curses.A_BOLD)
    scrn.nodelay(1) #Don't wait for a
keystroke...just cache it.
    self.ResetForNew()
    self.GameScore = 0
    self.Lives = 3
    self.PrintScore(scrn)
    scrn.move(self.ScoreLine,self.ScorePosition)
```

```
def PrintScore(self,scrn):

scrn.addstr(self.ScoreLine,self.ScorePosition,"SCORE:
%d" % self.GameScore)

scrn.addstr(self.ScoreLine,self.LivesPosition,"LIVES:
%d" % self.Lives)
```

PROGRAMMARE IN PYTHON - PARTE 14

forzare l'attributo a vero. Quindi dobbiamo dare corpo alla routine CheckKeys. Sostituite l'istruzione pass con il nuovo codice (pagina seguente, in basso a destra).

Ora abbiamo bisogno di una routine per muovere il proiettile verso l'alto (in basso a sinistra).

Abbiamo ancora bisogno di altre funzioni (pagina seguente, in alto a destra) prima di finire. Ecco il codice per completare la funzione CheckForHit e il codice per ExplodeBullet.

Per finire, completiamo Explode. Sostituiamo pass con il seguente codice (prossima pagina, in basso).

Abbiamo finalmente un programma funzionante. Potete personalizzare il valore in LetterWaitCount per velocizzare o rallentare il movimento della lettera sullo schermo per renderlo più difficile o facile. Potete anche usare la variabile CurrentColor per scegliere casualmente uno dei quattro colori da assegnare alla lettera. Vi lancia questa sfida.

Spero che vi siate divertiti questa volta e che aggiungerete altro codice per rendere il gioco più

divertente. Come al solito, l'intero progetto è disponibile presso www.thedesignedgeek.com o all'indirizzo <http://fullcirclemagazine.pastebin.com/DeReeh8m>

```
def MoveGun(self, scrn, direction):
    scrn.addch(self.GunLine, self.GunPosition, " ")
    if direction == 0: # left
        if self.GunPosition > 0:
            self.GunPosition -= 1
    elif direction == 1: # right
        if self.GunPosition < 79:
            self.GunPosition += 1

    scrn.addch(self.GunLine, self.GunPosition, self.GunChar,
               curses.color_pair(2) | curses.A_BOLD)
```

```
if keyin == 260: # left arrow - NOT on keypad
    self.MoveGun(scrn, 0)
    curses.flushinp() #Flush out the input buffer for safety.
elif keyin == 261: # right arrow - NOT on keypad
    self.MoveGun(scrn, 1)
    curses.flushinp() #Flush out the input buffer for safety.
elif keyin == 52: # left arrow ON keypad
    self.MoveGun(scrn, 0)
    curses.flushinp() #Flush out the input buffer for safety.
elif keyin == 54: # right arrow ON keypad
    self.MoveGun(scrn, 1)
    curses.flushinp() #Flush out the input buffer for safety.
elif keyin == 32: #space
    if self.Shooting == 0:
        self.Shooting = 1
        self.BulletColumn = self.GunPosition
        scrn.addch(self.BulletRow, self.BulletColumn, "|")
        curses.flushinp() #Flush out the input buffer for safety.
```

```
def MoveBullet(self, scrn):
    scrn.addch(self.BulletRow, self.BulletColumn, " ")
    if self.BulletRow > self.LetterLine:
        self.CheckForHit(scrn)
        self.BulletRow -= 1

    scrn.addch(self.BulletRow, self.BulletColumn, "|")
    else:
        self.CheckForHit(scrn)

    scrn.addch(self.BulletRow, self.BulletColumn, " ")
    self.BulletRow = self.GunLine - 1
    self.Shooting = 0
```



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.


```
def CheckForHit(self, scrn):
    if self.Shooting == 1:
        if self.BulletRow == self.CurrentLetterLine:
            if self.BulletColumn == self.CurrentLetterPosition:
                scrn.addch(self.BulletRow, self.BulletColumn, " ")

                self.ExplodeBullet(scrn)
                self.GameScore +=1
                self.ResetForNew()

def ExplodeBullet(self, scrn):
    scrn.addch(self.BulletRow, self.BulletColumn, "X", curses.color_pair(5))
    scrn.refresh()
    curses.napms(200)
    scrn.addch(self.BulletRow, self.BulletColumn, "|", curses.color_pair(5))
    scrn.refresh()
    curses.napms(200)
    scrn.addch(self.BulletRow, self.BulletColumn, "-", curses.color_pair(5))
    scrn.refresh()
    curses.napms(200)
    scrn.addch(self.BulletRow, self.BulletColumn, ".", curses.color_pair(5))
    scrn.refresh()
    curses.napms(200)
    scrn.addch(self.BulletRow, self.BulletColumn, " ", curses.color_pair(5))
    scrn.refresh()
    curses.napms(200)
```

```
scrn.addch(self.CurrentLetterLine, self.CurrentLetterPosition, "X", curses.color_pair(5))
curses.napms(100)
scrn.refresh()
scrn.addch(self.CurrentLetterLine, self.CurrentLetterPosition, "|", curses.color_pair(5))
curses.napms(100)
scrn.refresh()
scrn.addch(self.CurrentLetterLine, self.CurrentLetterPosition, "-", curses.color_pair(5))
curses.napms(100)
scrn.refresh()
scrn.addch(self.CurrentLetterLine, self.CurrentLetterPosition, ".", curses.color_pair(5))
curses.napms(100)
scrn.refresh()
scrn.addch(self.CurrentLetterLine, self.CurrentLetterPosition, " ")
scrn.addch(self.GunLine, self.GunPosition, self.GunChar, curses.color_pair(2) | curses.A_BOLD)
scrn.refresh()
```



Questo mese andremo ad esplorare Pygame, un set di moduli progettato per scrivere giochi. il sito web è

<http://www.pygame.org/>. Una citazione dal file readme di Pygame: "Pygame è una libreria multi-piattaforma progettata per rendere facile la scrittura di software multimediali, come i giochi, in Python. Pygame richiede il linguaggio Python e la libreria multimediale SDL. Può anche fare uso di altre diverse librerie molto note."

Si può installare Pygame attraverso Synaptic come 'python-game'. Fatelo ora in modo che possiamo andare avanti.

In primo luogo importiamo Pygame (vedi sopra a destra). Successivamente imposteremo `os.environ` per far sì che la nostra finestra sia centrata sul nostro schermo. Dopo di che inizieremo Pygame, poi imposteremo la finestra Pygame a 800x600 pixel e poi la didascalìa. Infine visualizzeremo la schermata e andremo in un loop di attesa della pressione di un tasto sulla tastiera o di un pulsante del mouse. Lo schermo è un oggetto che contiene qualunque cosa decidiamo di

infilarci. È chiamato superficie. Pensate come se fosse un pezzo di carta su cui disegneremo le cose. Non molto eccitante, ma è già un inizio. Rendiamolo un po' meno noioso. Possiamo cambiare il colore dello sfondo con un qualcosa di meno scuro. Ho trovato un programma di nome "colorname" che potete installare tramite Ubuntu Software Center. Questo vi permette di usare la "ruota dei colori" per prendere il colore che vi piace e vi darà i valori RGB ovvero Red, Green, Blue di quel colore. Dobbiamo usare i colori RGB se non vogliamo usare i colori predefiniti che Pygame ci offre. Si tratta di un programma accurato di cui si dovrebbe prendere in considerazione l'installazione. Subito dopo le dichiarazioni d'importazione aggiungete...

```
Background = 208, 202, 104
```

Questo imposterà la variabile Background su un colore dorato. Successivamente dopo la linea `pygame.display.set_caption`, aggiungete le seguenti linee...

```
screen.fill(Background)
pygame.display.update()
```

```
#This is the Import
import pygame
from pygame.locals import *
import os
# This will make our game window centered in the screen
os.environ['SDL_VIDEO_CENTERED'] = '1'
# Initialize pygame
pygame.init()
#setup the screen
screen = pygame.display.set_mode((800, 600))
# Set the caption (title bar of the window)
pygame.display.set_caption('Pygame Test #1')
# display the screen and wait for an event
doloop = 1
while doloop:
    if pygame.event.wait().type in (KEYDOWN,
    MOUSEBUTTONDOWN):
        break
```

Il metodo `screen.fill()` imposterà il colore a tutto ciò su cui passiamo sopra. La linea successiva, `pygame.display.update()`, in realtà aggiorna i cambiamenti ai nostri schermi.

Salvate questo con il nome di `pygame1.py` e andiamo avanti.

Ora mostreremo alcuni testi sul nostro modesto schermo. Nuovamente iniziamo con l'importare le nostre dichiarazioni e la variabile di assegnazione di background del nostro ultimo programma.

```
import pygame
from pygame.locals import *
import os
Background = 208, 202, 104
```

Ora, aggiungete una variabile aggiuntiva per il colore di primo piano della nostra font.

```
FontForeground = 255,255,255 #
White
```

Quindi aggiungeremo nella maggior parte del codice dal nostro ultimo programma (mostrato a destra).

Se lo si esegue adesso non è cambiato niente visivamente dal momento che abbiamo aggiunto la definizione in primo piano. Ora dopo la linea `screen.fill()` e prima del loop del nostro codice, inserite le seguenti linee:

```
font =  
pygame.font.Font(None,27) text  
= font.render('Here is some  
text', True, FontForeground,  
Background) textrect =  
text.get_rect()  
screen.blit(text, textrect)  
pygame.display.update()
```

Andate avanti, salvate il programma con il nome di `pygame2.py` ed eseguitelo. Nello Schermo in alto a sinistra si dovrebbe vedere il testo "Here is some text".

Andiamo a scrivere alcuni comandi. In primo luogo chiamiamo il metodo `Font` e lo passiamo a due argomenti. Il primo è il nome della font che desideriamo usare e il secondo è la dimensione della font. In questo momento ci limiteremo a usare 'None' e lasciare che il sistema scelga un tipo di carattere per noi e impostiamo la dimensione del carattere a 27 punti.

Dopo abbiamo il metodo `font.render()`. Questo ha quattro argomenti. In ordine ci sono: i testi

che desideriamo mostrare se vogliamo usare l'anti-aliasing (Vero in questo caso), il colore del carattere in primo piano e infine il colore del carattere di sfondo.

La linea successiva (`text.get_rect()`) assegna un oggetto rettangolare che useremo per inserire il testo sullo schermo. Questa è una cosa importante dal momento che quasi tutto il resto di cui ci occuperemo è con i rettangoli (capirete di più in un secondo momento). Poi muoviamo (blit) il rettangolo sullo schermo. E finalmente aggiorniamo lo schermo per far mostrare il nostro testo. Cosa vuol dire "blit" e perché diamine dovrei fare qualcosa che suona così strano? Il termine risale agli anni '70 e veniva da Xerox PARC (da cui proviene molta della tecnologia odierna). Il termine originale era BitBLT che significa Bit (or Bitmap) Block Transfer. Poi venne cambiato in Blit (forse perché è più corto). Praticamente stiamo muovendo la nostra immagine o il nostro testo sullo schermo.

Che fare se vogliamo che il testo venga centrato sullo schermo invece che sulla riga in alto dove ci vuole un po' di tempo per vederlo? Tra la linea `text.get_rect()` e `screen.blit` inserite le

```
# This will make our game window centered in the screen  
os.environ['SDL_VIDEO_CENTERED'] = '1'  
# Initialize pygame  
pygame.init()  
# Setup the screen  
screen = pygame.display.set_mode((800, 600))  
# Set the caption (title bar of the window)  
pygame.display.set_caption('Pygame Test #1')  
screen.fill(Background)  
pygame.display.update()  
  
# Our Loop  
doloop = 1  
while doloop:  
    if pygame.event.wait().type in (KEYDOWN,  
    MOUSEBUTTONDOWN):  
        break
```

seguenti due linee:

```
textRect.centerX =  
screen.get_rect().centerx  
textRect.centery =  
screen.get_rect().centery
```

Adesso stiamo rilevando il centro dell'oggetto schermo (ricordate, superficie) nelle posizioni in pixel x e y e stiamo impostando i punti centrali x e y del nostro oggetto `textRect` a quei valori.

Eseguite il programma. Ora il nostro testo è centrato nella superficie. Potete inoltre modificare il testo usando (nel nostro semplice codice) `font.set_bold(True)` e/o

`font.set_italic(True)` a destra dopo la linea `pygame.font.Font`.

Ricordate, abbiamo discusso brevemente sull'impostazione 'None' quando impostiamo il tipo di carattere a un font generico. Immaginiamo che di voler utilizzare un font più elaborato. Come ho detto prima il `pygame.font.Font()` method ha due argomenti. Il primo si riferisce al percorso e al nome del file della font che vorremmo usare, il secondo fa riferimento alla dimensione del carattere. A questo punto il problema è il percorso. Come facciamo a sapere il vero percorso e il nome del file della font che vorremmo usare in un qualunque sistema? Per fortuna

Pygame ha una funzione che ci pensa per noi. Si chiama `match_font`. Ecco qui un programma che mostrerà il percorso e il nome del file della font (in questo caso) Courier New.

```
import pygame
from pygame.locals import *
import os
print
pygame.font.match_font('Courier New')
```

Nel mio sistema il valore restituito è `"/usr/share/fonts/truetype/msttcorefonts/cour.ttf"`. Se però il font non viene trovato il valore di ritorno è `"None"`. Ammettendo il caso che la font È stata trovata, allora possiamo assegnare a una variabile il valore restituito e di conseguenza possiamo usare le seguenti attribuzioni.

```
courier =
pygame.font.match_font('Courier New') font =
pygame.font.Font(courier, 27)
```

Cambiate la vostra ultima versione del programma inserendo queste due linee e provate di nuovo. L'ultima linea dice questo: usate un carattere che voi SAPETE essere disponibile nel computer dell'utente finale oppure includetelo quando distribuite il vostro programma e codificate il

percorso e il nome della font. Ci sono altri modi per fare la stessa cosa, ma lo lascio capire a voi in modo che possiamo andare avanti.

Se il testo è bello, la grafica è però migliore. Ho trovato un tutorial veramente carino per Pygame scritto da Peyton McCollugh e ho pensato di prenderlo e modificarlo. Per questa parte abbiamo bisogno di iniziare con una figura che si muoverà intorno la nostra superficie. Questa figura è nota come 'sprite'. Utilizzate GIMP o qualche altro strumento per creare una figura stilizzata. Niente di fantastico, solo una generica figura stilizzata. Si presume che stiate usando GIMP. Create una nuova immagine, impostate le dimensioni di altezza e larghezza a 50 pixel e nelle opzioni avanzate impostate il riempimento su trasparente. Utilizzate lo strumento matita con un pennello ritondo (03). Disegnate la vostra piccola figura e salvatela come `stick.png` nella stessa cartella che avete usato per il codice. Ecco come appare quella mia. Sono sicuro che voi sapete fare di meglio.



Lo so... non sono un'artista. Tuttavia per i nostri scopi basta. Abbiamo salvato il file

```
import pygame
from pygame.locals import *
import os
```

```
Background = 0,255,127
os.environ['SDL_VIDEO_CENTERED'] = '1'
pygame.init()
screen = pygame.display.set_mode((800, 600))
pygame.display.set_caption('Pygame Example #4 - Sprite')
screen.fill(Background)
```

come `.png` e abbiamo impostato lo sfondo su trasparente in modo che vengano mostrate solo le piccole linee nere della nostra figura stilizzata e non uno sfondo bianco o di un altro colore.

Vediamo adesso cosa vogliamo che il programma faccia. Vogliamo mostrare una finestra di Pygame con la nostra figura stilizzata in essa. Vogliamo che la figura si muova quando premiamo qualunque freccia direzionale (su, giù, destra, sinistra) sempre che non siamo sul bordo dello schermo e la figura non si possa muovere ulteriormente. Vorremo uscire dal gioco quando premiamo il tasto "q". Ora spostare lo sprite in giro potrebbe sembrare facile e lo è, ma è un po' più difficile di quanto sembri inizialmente. Iniziamo creando due rettangoli. Uno per lo sprite se stesso e uno delle stesse dimensioni ma bianco. Muoviamo lo sprite sulla

superficie per iniziare, poi quando l'utente preme un tasto muoviamo il rettangolo bianco sopra lo sprite originale, rileviamo la nuova posizione e muoviamo nuovamente lo sprite sulla superficie nella nuova posizione. Più o meno quello che abbiamo fatto l'ultima volta con il gioco dell'alfabeto. Questo è tutto per questo programma. Ci darà un'idea di come mettere effettivamente un elemento grafico sullo schermo e muoverlo.

Quindi iniziamo un nuovo programma e lo chiamiamo `pygame4.py`. Posizionatelo nella cartella `includes` che abbiamo usato durante questo tutorial. Questa volta utilizzeremo uno sfondo di colore verde menta, quindi i valori dovrebbero essere `0, 255, 127` (vedi sopra).

Successivamente creiamo una classe che gestirà la nostra grafica o lo

PROGRAMMARE IN PYTHON - PARTE 15

sprite (è mostrato nella pagina successiva in basso a sinistra). Mettete questo parametro dopo le importazioni.

Che cos'è tutto questo che sto facendo? Cominciamo con la routine `__init__`. Inizializziamo il modulo dello sprite di Pygame con la linea `pygame.sprite.Sprite.__init__`. Poi impostiamo la superficie e la chiamiamo schermo. Questo ci

permetterà di controllare se lo 'sprite' va fuori dallo schermo.

Successivamente creiamo e impostiamo la posizione della variabile vuota `oldsprite`, che manterrà la posizione del nostro sprite. Ora carichiamo la nostra figura stilizzata con la routine `pygame.image.load`, indicandole il nome del file (e il percorso, se questo non è nel percorso del programma). Quindi acquisiamo un riferimento

(`self.rect`) allo sprite che imposti automaticamente la larghezza e l'altezza del rettangolo e imposti la posizione x,y di quel rettangolo alla posizione che abbiamo passato nella routine.

La routine di aggiornamento praticamente crea una copia dello sprite, poi controlla se questo va fuori dallo schermo. Se è così, rimane dov'è, altrimenti la sua posizione viene spostata di quel tanto che che gli

abbiamo indicato.

Ora, dopo la dichiarazione `screen.fill`, inserite il codice riportato nella pagina seguente (lato destro).

Qui creiamo un'istanza della nostra classe di nome "character". Poi muoviamo lo sprite. Creiamo lo sprite rettangolare vuoto e riempiamolo con il colore di sfondo. Aggiorniamo la superficie e iniziamo il nostro ciclo.

Fino a quando `DoLoop` è equivalente a 1, effettuiamo il loop attraverso il codice. Usiamo `pygame.event.get()` per avere un carattere dalla tastiera. Poi lo verifichiamo a seconda del tipo di evento. Se è `QUIT`, usciamo. Se è un evento `KEYDOWN` di pygame lo eseguiamo. Guardiamo il valore della chiave restituita e la confrontiamo con le costanti definite da Pygame. Poi chiamiamo l'aggiornamento della routine nella nostra classe. Notate qui che stiamo semplicemente passando una lista che contiene i numeri dei pixel degli assi X e Y per muovere il personaggio. Lo cambiamo di 10 pixel (positivo verso destra o sotto e negativo verso sinistra o sopra). Se il valore della chiave è uguale a "q", impostiamo `DoLoop` a 0 e usciamo dal loop. Dopo tutto questo muoviamo il personaggio vuoto alla vecchia

```
class Sprite(pygame.sprite.Sprite):
    def __init__(self, position):
        pygame.sprite.Sprite.__init__(self)
        # Save a copy of the screen's rectangle
        self.screen = pygame.display.get_surface().get_rect()
        # Create a variable to store the previous position of the sprite
        self.oldsprite = (0, 0, 0, 0)
        self.image = pygame.image.load('stick3.png')
        self.rect = self.image.get_rect()
        self.rect.x = position[0]
        self.rect.y = position[1]

    def update(self, amount):
        # Make a copy of the current rectangle for use in erasing
        self.oldsprite = self.rect
        # Move the rectangle by the specified amount
        self.rect = self.rect.move(amount)
        # Check to see if we are off the screen
        if self.rect.x < 0:
            self.rect.x = 0
        elif self.rect.x > (self.screen.width - self.rect.width):
            self.rect.x = self.screen.width - self.rect.width
        if self.rect.y < 0:
            self.rect.y = 0
        elif self.rect.y > (self.screen.height - self.rect.height):
            self.rect.y = self.screen.height - self.rect.height
```

posizione, muoviamo lo sprite nella nuova posizione e come ultima cosa aggiorniamo; ma in questo caso aggiorniamo solo i due rettangoli contenenti lo sprite vuoto e quello attivo. Questo consente di risparmiare una quantità enorme di tempo e di elaborazione.

Come sempre l'intero codice è raggiungibile al sito

www.thedesignatedgeek.com o

all'indirizzo

<http://fullcirclemagazine.pastebin.com/DvSpZbaj>.

Si possono fare molte più cose con Pygame. Consiglio di visitare il loro sito e di guardare la pagina di riferimento

(<http://www.pygame.org/docs/ref/index.html>). In aggiunta potete dare

un'occhiata ai giochi che gli altri hanno caricato.

La prossima volta scaveremo più in profondità su Pygame creando un gioco che proviene dal mio passato... dal mio passato molto LONTANO.



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.

```
character = Sprite((screen.get_rect().x, screen.get_rect().y))
screen.blit(character.image, character.rect)

# Create a Surface the size of our character
blank = pygame.Surface((character.rect.width, character.rect.height))
blank.fill(Background)

pygame.display.update()
DoLoop = 1
while DoLoop:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            sys.exit()
        # Check for movement
        elif event.type == pygame.KEYDOWN:
            if event.key == pygame.K_LEFT:
                character.update([-10, 0])
            elif event.key == pygame.K_UP:
                character.update([0, -10])
            elif event.key == pygame.K_RIGHT:
                character.update([10, 0])
            elif event.key == pygame.K_DOWN:
                character.update([0, 10])
            elif event.key == pygame.K_q:
                DoLoop = 0

# Erase the old position by putting our blank Surface on it
screen.blit(blank, character.oldsprite)
# Draw the new position
screen.blit(character.image, character.rect)
# Update ONLY the modified areas of the screen
pygame.display.update([character.oldsprite, character.rect])
```



Tempo fa promisi a qualcuno che avrei trattato le differenze tra Python 2.x e 3.x. La volta scorsa dissi che avremmo continuato la programmazione con pygame ma sento che dovrei mantenere la mia promessa così approfondiremo di più pygame la prossima volta.

In Python 3.x sono stati fatti molti cambiamenti. Sul web c'è una gran quantità di informazioni riguardo questi mutamenti e alla fine dell'articolo includerò alcuni collegamenti. Vi sono anche molte preoccupazioni relativamente al fare il cambiamento. Mi concentrerò sulle variazioni che riguardano le cose che avete imparato finora.

Forza, iniziamo.

PRINT

Come ho detto prima uno degli argomenti più importanti è il modo in cui affrontiamo il comando print. Con la versione 2.x possiamo usare semplicemente:

```
print "This is a test"
```

e così sarà fatto. Tuttavia, con la 3.x, se ci proviamo otterremo il messaggio di errore mostrato sopra a destra.

Non è bello. Per usare il comando print dobbiamo mettere ciò che vogliamo stampare tra parentesi tonde così:

```
print ("This is a test")
```

Non è un cambiamento molto grande ma è qualcosa di cui dobbiamo essere consapevoli. Potete prepararvi alla migrazione utilizzando questa sintassi sotto Python 2.x.

Formattazione e sostituzione di variabile

Anche la formattazione e la sostituzione di variabile sono cambiate. Con la versione 2.x abbiamo usato cose simili all'esempio mostrato sotto a sinistra e, con la versione 3.1, potete ottenere il giusto risultato. Comunque ciò è dovuto al cambiamento dato che le funzioni

```
>>> print "This is a test"
File "<stdin>", line 1
    print "This is a test"
      ^
SyntaxError: invalid syntax
>>>
```

di formattazione '%s' e '%d' spariranno. Il nuovo modo, mostrato sotto, è usare le dichiarazioni di sostituzione '{x}'.

In effetti mi sembra essere più facile da leggere. Potete anche fare cose come questa:

```
>>> print ("Hello {0}. I'm glad you are here at {1}.format("Fred", "MySite.com"))

Hello Fred. I'm glad you are here at MySite.com

>>>
```

Ricordate, potete ancora usare '%s' e così via ma essi spariranno.

Numeri

Sotto Python 2.x, se facevate:

```
x = 5/2.0
```

x avrebbe contenuto 2.5. Tuttavia se aveste fatto:

```
>>> months = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']
>>> print "You selected month %s" % months[3]
You selected month Apr
>>>
```

VECCHIO METODO

```
>>> months = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']
>>> print("You selected month {0}".format(months[3]))
You selected month Apr
>>>
```

NUOVO METODO

```
x = 5/2
x avrebbe contenuto 2 grazie al
troncamento. Sotto la versione 3.x
se fate:
x = 5/2
ottenete ancora 2.5. Per troncare la
divisione dovete fare:
x = 5//2
```

Input

Un po' di tempo fa abbiamo avuto a che fare con un sistema di menù che usava `raw_input()` per ottenere una risposta dall'utente della nostra applicazione. Qualcosa che andava così:

```
response = raw_input('Enter a
selection -> ')
```

Questo andava bene sotto la versione 2.x. Tuttavia sotto la 3.x otteniamo:

```
Traceback (most recent call
last):
```

```
File "<Stdin>", line 1,
in <module>
```

```
NameError: name 'raw_input'
is not defined
```

Questo non è un grosso problema. Il metodo `raw_input()` è stato sostituito da `input()`. Semplicemente, cambiate la riga in:

```
response = input ('Enter a
selection -> ')
```

e funziona proprio bene.

Non uguale

Sotto la versione 2.x avremmo potuto fare un test di "non uguaglianza" con "`<>`". Tuttavia ciò non è consentito nella versione 3.x. L'operatore di prova adesso è "`!=`".

Convertire i programmi più vecchi in Python 3.x

```
#pprint1.py
#Example of semi-useful functions

def TopOrBottom(character,width):
    # width is total width of returned line
    return '%s%s%s' % ('+',(character * (width-2)),'+')

def Fmt(val1,leftbit,val2,rightbit):
    # prints two values padded with spaces
    # val1 is thing to print on left, val2 is thing to print on right
    # leftbit is width of left portion, rightbit is width of right portion
    part2 = '%.2f' % val2
    return '%s%s%s%s' % ('| ',val1.ljust(leftbit-2,' '),part2.rjust(rightbit-2,' '),
|')
# Define the prices of each item
item1 = 3.00
item2 = 15.00
# Now print everything out...
print TopOrBottom('= ',40)
print Fmt('Item 1 ',30,item1,10)
print Fmt('Item 2 ',30,item2,10)
print TopOrBottom('- ',40)
print Fmt('Total ',30,item1+item2,10)
print TopOrBottom('= ',40)
```

Python 3.x arriva con una utility che aiuta a convertire un'applicazione 2.x in codice conforme alla versione 3.x. Non funziona

sempre ma vi ci porterà vicini in molti casi. Lo strumento di conversione viene chiamato "2to3". Prendiamo come esempio un programma davvero semplice. L'esempio sotto è preso da *Programmazione in Python Parte 3* di tempo addietro.

```
+=====+
| Item 1           3.00 |
| Item 2          15.00 |
+-----+
| Total           18.00 |
+=====+
Script terminated.
```

Quando viene eseguito sotto la versione 2.x, l'output è simile a quello mostrato sopra a destra.

Naturalmente quando lo eseguiamo sotto la 3.x non funziona.


```
File "pprint1.py", line 18
    print TopOrBottom('=' ,40)
```

SyntaxError: invalid syntax

Proveremo a lasciare che l'applicazione di conversione lo sistemi per noi. Per prima cosa dovremmo creare una copia di riserva dell'applicazione che sarà convertita. Io lo faccio creando una copia del file e aggiungendo un "v3" alla fine del nome:

```
cp pprint1.py pprintlv3.py
```

Vi sono molteplici modi di eseguire l'applicazione. Il modo più semplice è lasciare che l'applicazione controlli il codice e ci dica dove sono i problemi, il che viene mostrato sotto a sinistra.

Notate che il codice sorgente originale non è cambiato. Dobbiamo usare il flag "-w" per dirgli di scrivere sul file i

cambiamenti. Ciò è mostrato sotto a destra.

Noterete anche che l'output è identico. Questa volta, comunque, il file sorgente (mostrato nella pagina successiva) è cambiato in un file "versione 3 compatibile".

Adesso il programma funziona come dovrebbe sotto la versione 3.x. E, dato che era semplice, funziona ancora anche sotto la

versione 2.x.

Passo adesso alla versione 3.x?

Molti dei problemi sono comuni a qualunque cambiamento in un linguaggio di programmazione. I cambiamenti di sintassi abbondano ad ogni nuova versione. A volte spuntano fuori dal nulla scorciatoie come += o -= e rendono la nostra

```
> 2to3 pprintlv3.py
RefactoringTool: Skipping implicit fixer: buffer
RefactoringTool: Skipping implicit fixer: idioms
RefactoringTool: Skipping implicit fixer: set_literal
RefactoringTool: Skipping implicit fixer: ws_comma
RefactoringTool: Refactored pprintlv3.py
--- pprintlv3.py (original)
+++ pprintlv3.py (refactored)
@@ -15,9 +15,9 @@
     item1 = 3.00
     item2 = 15.00
     # Now print everything out...
-print TopOrBottom('=' ,40)
-print Fmt('Item 1',30,item1,10)
-print Fmt('Item 2',30,item2,10)
-print TopOrBottom('-',40)
-print Fmt('Total',30,item1+item2,10)
-print TopOrBottom('=' ,40)
+print(TopOrBottom('=' ,40))
+print(Fmt('Item 1',30,item1,10))
+print(Fmt('Item 2',30,item2,10))
+print(TopOrBottom('-',40))
+print(Fmt('Total',30,item1+item2,10))
+print(TopOrBottom('=' ,40))
RefactoringTool: Files that need to be modified:
RefactoringTool: pprintlv3.py
```

```
> 2to3 -w pprintlv3.py
RefactoringTool: Skipping implicit fixer: buffer
RefactoringTool: Skipping implicit fixer: idioms
RefactoringTool: Skipping implicit fixer: set_literal
RefactoringTool: Skipping implicit fixer: ws_comma
RefactoringTool: Refactored pprintlv3.py
--- pprintlv3.py (original)
+++ pprintlv3.py (refactored)
@@ -15,9 +15,9 @@
     item1 = 3.00
     item2 = 15.00
     # Now print everything out...
-print TopOrBottom('=' ,40)
-print Fmt('Item 1',30,item1,10)
-print Fmt('Item 2',30,item2,10)
-print TopOrBottom('-',40)
-print Fmt('Total',30,item1+item2,10)
-print TopOrBottom('=' ,40)
+print(TopOrBottom('=' ,40))
+print(Fmt('Item 1',30,item1,10))
+print(Fmt('Item 2',30,item2,10))
+print(TopOrBottom('-',40))
+print(Fmt('Total',30,item1+item2,10))
+print(TopOrBottom('=' ,40))
RefactoringTool: Files that were modified:
RefactoringTool: pprintlv3.py
```



vita più facile, in effetti.

Quale è lo svantaggio del migrare semplicemente alla versione 3.x proprio adesso? Beh, ce n'è un po'. Molti moduli di librerie che abbiamo utilizzato non sono disponibili per la versione 3 proprio adesso. Cose come Mutegen, che abbiamo utilizzato qualche articolo fa, non sono proprio ancora disponibili. Quantunque questo sia un ostacolo, non richiede che rinunciate completamente a Python 3.x.

Il mio suggerimento è di cominciare ora a scrivere codice utilizzando un'apposita sintassi 3.x. La versione 2.6 di Python supporta quasi tutto ciò di cui avreste bisogno per scrivere in modalità 3.x. In questo modo sarete pronti a partire una volta che dovrete cambiare alla 3.x. Se riuscite a sopravvivere con la libreria di moduli standard, continuate e fate il salto. Se, d'altro canto, andate oltre i limiti potreste voler attendere fino a che la libreria dei moduli si aggiorna. E lo farà.

Sotto vi sono alcuni collegamenti che ho pensato potessero essere utili. Il primo è alla pagina sull'impiego di 2to3. Il

```
#pprint1.py
#Example of semi-useful functions

def TopOrBottom(character,width):
    # width is total width of returned line
    return '%s%s%s' % ('+',(character * (width-2)),'+')
def Fmt(val1,leftbit,val2,rightbit):
    # prints two values padded with spaces
    # val1 is thing to print on left, val2 is thing to print on right
    # leftbit is width of left portion, rightbit is width of right portion
    part2 = '%.2f' % val2
    return '%s%s%s%s' % ('| ',val1.ljust(leftbit-2,' '),part2.rjust(rightbit-2,' '),
|')
# Define the prices of each item
item1 = 3.00
item2 = 15.00
# Now print everything out...
print(TopOrBottom('=' ,40))
print(Fmt('Item 1',30,item1,10))
print(Fmt('Item 2',30,item2,10))
print(TopOrBottom('-' ,40))
print(Fmt('Total',30,item1+item2,10))
print(TopOrBottom('=' ,40))
```

secondo è un bignamino di 4 pagine che ho scoperto essere un riferimento molto buono. Il terzo è a ciò che considero essere il miglior libro sull'utilizzo di Python (questo fino a che deciderò di scrivere il mio).

Arrivederci alla prossima volta.

Collegamenti

Utilizzo di 2to3
<http://docs.python.org/library/2to3.html>

Passare da Python 2 a Python 3 (un bignamino di 4 pagine)
http://ptqmedia.pearsoncmg.com/imprint_downloads/informit/promotions/python/python2python3.pdf

Dive into Python 3
<http://diveintopython3.org>



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.

