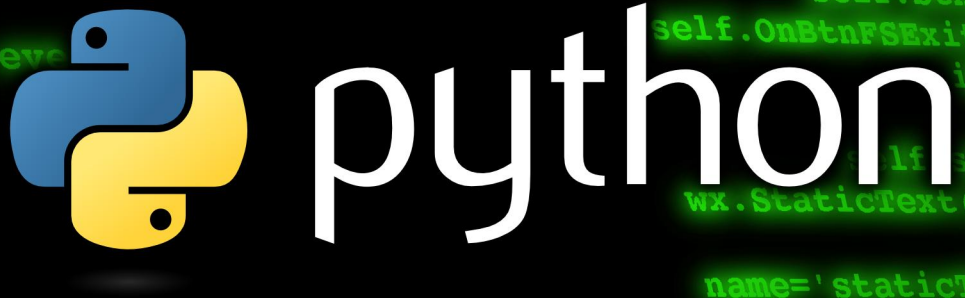
full circle

LA RIVISTA INDIPENDENTE PER LA COMUNITÀ LINUX UBUNTU

EDIZIONE SPECIALE SERIE PROGRAMMAZIONE

EDIZIONE SPECIALE
SERIE PROGRAMMAZIONE



python™

PROGRAMMARE IN PYTHON VOLUME 1

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

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

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

Full Circle

LA RIVISTA INDIPENDENTE PER LA COMUNITÀ LINUX UBUNTU

Ecco a voi un altro Speciale monotematico!

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

Quella che avete davanti è la ristampa della serie **Programmare in Python**, **parti 1-8**, pubblicata nei numeri 27-34: niente di speciale, giusto quello che abbiamo già pubblicato.

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

Buon divertimento!

Come contattarci
Sito web:
http://www.fullcirclemagazine.org/
Forum:
http://ubuntuforums.org/forumdisplay.php?f=270
IRC:
#fullcirclemagazine su chat.freenode.net

Gruppo editoriale
Capo redattore: Ronnie Tucker (aka: RonnieTucker)
ronnie@fullcirclemagazine.org
Webmaster: Rob Kerfia (aka: admin / linuxgeekery-
admin@fullcirclemagazine.org
Podcaster: Robin Catling (aka RobinCatling)
podcast@fullcirclemagazine.org
Manager delle comunicazioni: Robert Clipsham (aka: mrmonday) -
mrmonday@fullcirclemagazine.org

Programmare In Python - Parte 1

Scritto da Greg Walters

VEDI ANCHE:
N/A

DISPONIBILE PER:
ubuntu kubuntu xubuntu

CATEGORIE:
Sviluppo Grafica Internet M/media Sistema

DISPOSITIVI:
CD/DVD HDD USB Drive Laptop Wireless

Tra i tanti linguaggi di programmazione disponibili oggi, Python è quello di più facile apprendimento. Python vede la luce verso la fine degli anni 80 e, a partire da allora, è enormemente maturato. Solitamente si trova già installato nella maggior parte delle distribuzioni Linux ed è spesso il linguaggio di riferimento per chi intende imparare a programmare. In questo articolo affronteremo la programmazione da riga di comando, in un prossimo articolo quella con GUI (Graphical User Interface, ndt Interfaccia Utente Grafica). Iniziamo subito con la creazione di una semplice applicazione.

Il nostro primo programma

Utilizzando un editor di testo come gedit, scriviamo qualche riga di codice. Scrivete le seguenti quattro righe:

```
#!/usr/bin/env python

print 'Ciao. Sono un programma in python.'

nome = raw_input("Qual'è il tuo nome? ")

print "Benvenuto " + nome + "!"
```

E questo è tutto ciò che ci occorre. Salvate il file con il nome di ciao.py in qualsiasi posizione preferiate. Suggerirei di utilizzare una cartella di nome esempi_python da creare nella propria home. Questo piccolo esempio mostra come sia semplice scrivere codice in Python. Prima di eseguire il programma dobbiamo impostare il file come eseguibile. Lo faremo con il seguente comando:

```
chmod +x ciao.py
```

eseguito all'interno della cartella in cui avete salvato il file python. A questo punto, eseguiamo il programma.

```
greg@earth:~/esempi_python$ ./ciao.py

Ciao. Sono un programma in python.
Quale è il tuo nome? Ferd Burphel
Benvenuto Fred Burphel!
greg@earth:~/esempi_python$
```

È stato semplice! Vediamo cosa faccia effettivamente ogni singola riga del nostro codice.

```
#!/usr/bin/env python
```

Questa riga dice al sistema che sta avendo a che fare con un programma python e quindi di utilizzare l'interprete python di default per eseguire il programma.

```
print 'Ciao. Sono un programma in python.'
```

In breve, questo stampa la prima riga 'Ciao. Sono un programma in python.' sul terminale.

```
nome = raw_input("Qual'è il tuo nome? ")
```

Questa è un pò più complessa. Ci sono due parti in questa riga. La prima è nome =, mentre la seconda è raw_input("Qual'è il tuo nome? "). Cominciamo a considerare la seconda parte. Il comando raw_input stamperà a video la frase "Qual'è il tuo nome? " e sucessivamente rimarrà in attesa che l'utente (voi) scriva qualcosa (seguito poi dal tasto {Invio}). Vediamo ora la prima parte: nome =. Questa parte del comando crea una variabile chiamata "nome". Cosa è una variabile ? Potete pensare ad una variabile come ad una scatola di cartone. Potete riporci dentro delle cose: scarpe, parti di computer, carta o qualsiasi altra cosa. Alla scatola non interessa cosa le si mette dentro: semplicemente lo conserverà. In questo caso, conserverà ciò che verra scritto. Nel mio esempio, ho scritto Ferd Burphel. Python prenderà la stringa e la

conserverà nella scatola chiamata "nome" in modo da utilizzarla sucessivamente nel programma.

```
print "Benvenuto " + nome + "!"
```

Usiamo ancora il comando print per mostrare qualcosa sullo schermo: in questo caso "Benvenuto " più qualsiasi cosa sia contenuto nella variabile "nome" seguito da un punto esclamativo. Abbiamo concatenato, o meglio, messo insieme tre pezzi di informazione: "Benvenuto ", l'informazione contenuta nella variabile "nome" e il punto esclamativo.

Prima di procedere con il prossimo esempio, approfondiamo alcuni argomenti. Apriamo un terminale e scriviamo:

```
python
```

Dovreste vedere qualcosa di simile al seguente:

```
greg@earth:~/esempi_python$
python

Python 2.5.2 (r252:60911, Oct
5 2008, 19:24:49)

[GCC 4.3.2] on linux2

Type "help", "copyright",
"credits" or "license" for
more information.

>>>
```

Vi trovate, a questo punto nella shell di python. A partire da qui potete eseguire un enorme numero di cose ma vediamo, prima di tutto, con cosa abbiamo a che fare. La prima cosa che potrete notare è la versione di python: la mia è 2.5.2. Sucessivamente, potrete notare una frase che dice che, per ottenere aiuto, potete scrivere "help" sul terminale. Lascerò che lo facciate autonomamente. Adesso scrivete:

```
print 2+2
```

e premete invio. Vi verrà risposto

```
>>> print 2+2
4
>>>
```

Notate che abbiamo scritto la parola "print" in minuscolo. Cosa sarebbe successo se avessimo scritto "Print 2+2"? la risposta dell'interprete sarebbe stata questa:

```
>>> Print 2+2
  File "<stdin>", line 1
    Print 2+2
          ^
SyntaxError: invalid syntax
>>>
```

Questo è dovuto al fatto che la parola "print" è riconosciuto come comando mentre "Print" non lo è. La differenza tra maiuscolo e minuscolo è molto importante in Python.

Adesso giochiamo un altro po' con le variabili.

```
var = 2+2
```

Noterete che non succede nulla a parte il fatto che Python vi riproporrà il solito prompt ">>>". Va tutto bene. Ciò che abbiamo detto a Python di fare è di creare una variabile (scatola) chiamata var e di metterci dentro la somma "2+2". Per vedere il contenuto della variabile var, scrivete:

```
print var
```

e premete invio

```
>>> print var
4
>>>
```

D'ora in poi potremo usare più e più volte var esattamente come se fosse il numero 4, come nel caso seguente:

```
>>> print var * 2
8
>>>
```

Se scriviamo nuovamente "print var" ecco cosa otterremo:

```
>>> print var
4
>>>
```

var non è cambiata. È ancora la somma di 2+2 ovvero 4.

Stiamo facendo delle cose molto semplici appositamente per un tutorial dedicato a chi è alle prime armi. La complessità andrà crescendo nei prossimi tutorial. Ma per il momento vediamo qualche altro esempio sulle variabili.

Scrivete, nell'inteprete, quanto segue:

```
>>> strng = 'È venuto il tempo
per tutti i buoni uomini di
venire in aiuto del partito!'
>>> print strng
È venuto il tempo per tutti i
buoni uomini di venire in
aiuto del partito!
>>>
```

Abbiamo creato una variabile di nome strng (diminutivo di

stringa) che contiene il valore "È venuto il tempo per tutti i buoni uomini di venire in aiuto del partito!". D'ora in poi (fino a che rimaniamo nella stessa istanza dell'interprete) la nostra variabile strng sarà sempre uguale, a meno che non decidiamo di cambiarla. Cosa succederebbe se decidessimo di moltiplicare questa variabile per 4?

```
>>> print strng * 4

È venuto il tempo per tutti i buoni uomini di venire in aiuto del partito!È venuto il tempo per tutti i buoni uomini di venire in aiuto del partito!È venuto il tempo per tutti i buoni uomini di venire in aiuto del partito!È venuto il tempo per tutti i buoni uomini di venire in aiuto del partito!

>>>
```

Beh, non è esattamente ciò che vi sareste aspettati, vero? Viene stampato il valore di strng 4 volte. Perchè? Beh, l'interprete sa che strng conteneva una stringa di caratteri, non un valore. Non è possibile fare matematica con le stringhe.

Cosa succederebbe se avessimo una variabile chiamata s contenente il carattere '4' come qui di seguito?

```
>>> s = '4'
>>> print s
4
```

Sembrerebbe che s contenga il valore intero 4, ma in realtà non è così. In realtà contiene la rappresentazione testuale di 4. Quindi, se proviamo a scrivere 'print s * 4' ecco cosa otterremmo:

```
>>> print s * 4
4444
>>>
```

Ancora una volta, l'interprete sa che s è una stringa e non un valore numerico. Lo sa perchè abbiamo racchiuso il numero 4 tra apici singoli, rendendolo una stringa.

Possiamo dimostrarlo scrivendo print type(s) per vedere cosa il sistema ritenga sia la variabile.

```
>>> print type(s)
<type 'str'>
>>>
```

È confermato. È un tipo stringa. Se volessimo utilizzarlo come valore numerico, dovremmo fare la seguente cosa:

```
>>> print int(s) * 4
16
>>>
```

La stringa (s), che è '4', è stata convertita in un intero e quindi moltiplicata per 4 che ci da 16.

Abbiamo introdotto il comando print, il comando raw_input, l'assegnazione delle variabili e la differenza tra stringhe e interi.

Andiamo ancora un pò più in là. Scrivete, nell'interprete Python, quit() ritrovandovi nuovamente al prompt dei comandi.

Semplice ciclo For

Proviamo, a questo punto, a programmare un semplice ciclo. Riaprite l'editor e scrivete il seguente programma.

```
#! /usr/bin/env python

for cntr in range(0,10):

    print cntr
```

Assicuratevi di usare il tasto tab prima di "print cntr". È fondamentale. Python, diversamente da altri linguaggi di programmazione, non utilizza le parentesi tonde "(" o graffe "{" per realizzare gruppi di codice, ma utilizza l'indentazione.

Salvate il programma col nome "ciclo_for.py". Prima di provare l'esecuzione, approfondiamo il concetto del ciclo for.

Un ciclo è una parte di codice che esegue un insieme di istruzioni per un determinato numero di volte. Nel nostro caso, il programma effettuerà 10 cicli, stampando il valore della variabile cntr (diminutivo di contatore). Il comanda espresso in italiano corrente sarebbe "assegna alla variabile cntr 0; per 10 volte stampa il contenuto di cntr; aumenta di uno cntr; ricomincia da capo". Sembra abbastanza semplice. La parte di codice "range(0,10)" dice di iniziare con il valore 0 e di continuare il ciclo fino a che il valore di cntr non sia 10 per poi interrompere il ciclo stesso.

Ora, come visto in precedenza, diamo il comando

```
chmod +x ciclo_for.py
```

ed eseguiamo il programma

```
./ciclo_for.py
```

in un terminale.

```
greg@earth:~/esempi_python$
./ciclo_for.py
0
1
2
3
4
5
6
7
8
9
greg@earth:~/esempi_python$
```

Bene! Sembra che tutto abbia funzionato anche se il conteggio è proseguito solo fino a 9 prima di fermarsi. Guardate nuovamente l'output. Ci sono 10 numeri stampati, partendo dallo 0 e arrivando al 9. Questò è ciò che abbiamo chiesto: stampa 10 volte il valore di cntr aggiungendo ogni volta uno alla variabile e poi termina non appena questo valore è 10.

Potete notare che, per quanto programmare può essere semplice, alle volte può divenire complesso e dovete essere sempre certi di quel che state chiedendo di fare al sistema. Se aveste modificato la chiamata in "range(1,10)" il ciclo avrebbe iniziato a contare da 1 ma si sarebbe comunque fermato a 9 poichè, non appena cntr avesse contenuto 10, il ciclo sarebbe terminato. Perciò, per fare in modo che venga stampato "1,2,3,4,5,6,7,8,9,10" dovremo usare "range(1,11)" perchè il ciclo for si interromperà non appena il numero maggiore viene raggiunto.

Notate anche la sintassi del comando. È "for variabile in range(valore iniziale, valore finale):". Il simbolo ":" avvisa che quel che segue sarà un blocco di codice che dovrà essere indentato. È estremamente importante che vi ricordiate di utilizzare i due punti ":" e che indentiate il codice contenuto nel blocco.

Se modifichiamo il codice nel seguente modo:

```
#! /usr/bin/env python

for cntr in range(1,11):
    print cntr
print 'Tutto fatto'
```

avremo il seguente output...

```
greg@earth:~/esempi_python$
./ciclo_for.py
1
2
3
4
5
6
7
8
9
10
Tutto fatto
greg@earth:~/esempi_python$
```

Assicuratevi che l'indentazione sia corretta. Ricordate: l'indentazione mostra la formattazione del blocco. Approfondiremo ulteriormente l'indentazione nei prossimi tutorial.

Questo è tutto per ora. La prossima volta faremo un ripasso prima di andare avanti con altre istruzioni per la programmazione python. Nel frattempo considerate l'installazione di un qualche editor specifico per python come Dr. Python o SPE (Stani's Python Editor) entrambi disponibili tramite Synaptic.

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

Programmare in Python - Parte 2

Scritto da Greg Walters

VEDI ANCHE:
FCM n. 27 - Python Parte 1

VALIDO PER:
ubuntu kubuntu xubuntu

CATEGORIE:
Sviluppo Grafica Internet M/media Sistema

DISPOSITIVI:
CD/DVD HDD USB Drive Laptop Wireless

Correzioni alla parte 1
Ho ricevuto un mail da parte di David Turner il quale mi ha fatto notare come l'utilizzo del tasto Tab per l'indentazione del codice sia fuorviante poiché alcuni editor potrebbero utilizzare più o meno di quattro spazi per effettuarla. E ciò è corretto. Molti programmatori Python (me compreso), per risparmiare tempo configurano il tasto Tab del proprio editor in modo che rappresenti quattro spazi. Il problema comunque è che, se qualcun'altro non ha nel suo editor le stesse configurazioni, si potrebbero creare dei problemi nel codice sia di funzionamento che di leggibilità. Ragion per cui, cercate di abituarvi ad utilizzare gli spazi al posto del tasto Tab.

Nella scorsa puntata abbiamo visto un programma elementare che utilizzava raw_input per ricevere una risposta dall'utente, alcuni semplici tipi di variabile e un semplice ciclo che utilizzava la dichiarazione "for". In questa puntata approfondiremo ulteriormente le variabili e scriveremo alcuni programmi.

Le liste

Analizziamo un nuovo tipo di variabili chiamate liste. In altri linguaggi una lista verrebbe considerata come un array. Tornando alla nostra analogia con le scatole di cartone, un array (o lista) sarebbe una serie di scatole, contenenti oggetti, incollate tra di loro. Per esempio, potremmo conservare forchette in una scatola, coltelli in un'altra e cucchiai in un'altra ancora. Vediamo una semplice lista dei nomi dei mesi. Il nostro codice verrebbe così:

```
mesi = ['Gen', 'Feb', 'Mar',
'Apr', 'Mag', 'Giu', 'Lug',
'Ago', 'Set', 'Ott', 'Nov',
'Dic']
```

Per creare una lista, racchiudiamo i valori tra parentesi quadre ('[' e ']'). Abbiamo chiamato la nostra lista 'mesi'. Per utilizzarla, dovremmo scrivere qualcosa tipo print mesi[0] o mesi[1] (comando che stamperebbe 'Gen' o 'Feb'). Ricordate di contare sempre partendo dallo zero. Per sapere la dimensione di una lista possiamo usare il comando:

```
print len(mesi)
```

che restituirà 12.

Un altro esempio di lista potrebbe essere quello delle categorie all'interno di un libro di cucina. Ad esempio:

```
categorie = ['Primi piatti',
'Carni', 'Pesce', 'Zuppe',
'Biscotti']
```

Quindi la categoria[0] sarà 'Primi piatti' mentre la categoria[4] sarà 'Biscotti'. Molto semplice. Sicuramente vi saranno venuti in mente diversi utilizzi per le liste.

Fino ad ora, abbiamo creato delle liste utilizzando informazioni sotto forma di stringhe. È possibile utilizzare anche numeri interi. Rifacendoci alla precedente lista dei mesi, possiamo creare una lista che contenga il numero di giorni per ognuno di loro.

```
GiorniInMese = [31, 28, 31, 30,
31, 30, 31, 31, 30, 31, 30, 31]
```

Se scrivessimo GiorniInMese[1] (per Febbraio) ci verrebbe restituito il numero intero 28. Notate che ho chiamato la lista GiorniInMese. Altrettanto facilmente avrei potuto chiamarla 'giorniinmese' o semplicemente 'X'... ma queste forme sarebbero meno leggibili. Una buona pratica di programmazione suggerisce (e ciò è comunque soggetto ad interpretazione) che i nomi delle variabili siano facilmente comprensibili. Analizzeremo più avanti il perché. Per ora, giochiamo un altro po' con le liste.

Prima di iniziare con il prossimo programma di esempio, vediamo alcune altre cose di Python.

Ancora sulle stringhe

Abbiamo brevemente introdotto le stringhe nella Parte 1 e ora le approfondiremo. Una stringa è una serie di caratteri. Nient'altro che questo. In effetti, potete pensare ad una stringa come un array di caratteri. Ad esempio, se assegniamo la stringa "Il tempo è giunto" ad una variabile di nome strng e successivamente vogliamo sapere quale sia il secondo carattere della stringa, ecco cosa potremmo scrivere:

```
strng = "Il tempo è giunto"
print strng[1]
```

Il risultato sarà 'l'. Ricordando che è sempre necessario iniziare a contare da 0, il primo carattere sarà [0], il secondo [1], il terzo [2] e così via. Se volessimo avere i caratteri a partire dalla posizione 3 fino alla posizione 9, dovremmo scrivere:

```
print strng[3:9]
```

che ci restituirà 'tempo'. Come nel ciclo for della volta scorsa, il conteggio si fermerà a 9 ma non restituirà il nono carattere, che sarebbe lo spazio dopo la parola 'tempo'.

Possiamo sapere la dimensione di una stringa utilizzando la funzione len():

```
print len(strng)
```

che restituirà 18. Se, nel nostro codice, volessimo scoprire in quale posizione, all'interno della stringa, si trova la parola 'tempo' possiamo scrivere:

```
pos = strng.find('tempo')
```

A questo punto la variabile pos (diminutivo di posizione) conterrà 3, che significa che la parola 'tempo' inizia alla posizione 3 della nostra stringa. Se avessimo chiesto alla nostra stringa, tramite la funzione find, di trovare una parola che non è contenuta nella stessa:

```
pos = strng.find('mela')
```

il valore contenuto in pos sarebbe -1.

Possiamo anche ottenere ogni singola parola contenuta nella stringa tramite il comando split. Per dividere la stringa, nelle componenti separate dallo spazio, diamo il comando:

```
print strng.split(' ')
```

che ci restituirà una lista contenente ['Il', 'tempo', "e'", 'giunto']. Questi comandi sono estremamente potenti. Ci sono numerose funzioni presenti in python che analizzeremo in seguito.

Sostituzione di stringhe

C'è un'ultima cosa di cui vorrei parlare prima di passare al nostro prossimo programma di esempio. Quando voglio stampare qualcosa che contenga sia caratteri che variabili di tipo testo, possiamo utilizzare quello che viene definito Sostituzione di Variabili. Lo si fa in modo molto semplice. Se volessimo sostituire una stringa, utilizzeremo il simbolo '%s' e successivamente diremo a Python con cosa sostituirlo. Per esempio, per stampare un mese dalla nostra precedente lista, potremmo utilizzare il seguente costrutto:

```
print 'Mese = %s' % mesi[0]
```

Il risultato sarà 'Mese = Gen'. Se volessimo sostituire un intero, utilizzeremmo il '%d'. Eccone un esempio:

```
Mesi = ['Gen', 'Feb', 'Mar',
'Apr', 'Mag', 'Giu', 'Lug',
'Ago', 'Set', 'Ott', 'Nov',
'Dic']
GiorniInMese = [31, 28, 31, 30,
31, 30, 31, 31, 30, 31, 30, 31]
for cntr in range(0,12):
  print '%s ha %d giorni.' % (
Mesi[cntr], GiorniInMese[cntr] )
```

Il risultato sarà:

```
Gen ha 31 giorni.
Feb ha 28 giorni.
Mar ha 31 giorni.
Apr ha 30 giorni.
Mag ha 31 giorni.
Giu ha 30 giorni.
Lug ha 31 giorni.
Ago ha 31 giorni.
Set ha 30 giorni.
Ott ha 31 giorni.
Nov ha 30 giorni.
Dic ha 31 giorni.
```

Una cosa molto importante da comprendere è l'utilizzo dell'apice singolo e dell'apice doppio. Sia che assegniamo una stringa ad una variabile nel seguente modo:

```
st = 'Il tempo è giunto'
```

che in quest'altro modo

```
st = "Il tempo è giunto"
```

il risultato sarà lo stesso.

Però, se è necessario inserire un apice singolo all'interno della stringa, come in questo caso:

```
st = 'Ha detto che l'ha visto'
```

vi verrà restituito un errore di sintassi. Sarà perciò necessario effettuare l'assegnazione nel seguente modo:

```
st = "Ha detto che l'ha visto"
```

Potete vederla così: per definire una stringa, questa deve essere contenuta tra un qualche tipo di apice (uno all'inizio e uno alla fine) i quali devono coincidere. Se vi trovate nel caso in cui è necessario avere diversi tipi di apice, utilizzate come apici esterni quelli che non vi sono necessari all'interno della stringa, come nell'esempio precedente. Vi chiederete: e se dovessi inserire una stringa del tipo "Mi disse "Non c'è problema"" ? In questo caso, potrete definire la stringa nel modo seguente:

```
st = 'Mi disse "Non c\'è problema"'
```

Noterete il segno di backslash davanti al simbolo di apice singolo in 'Non c'è'. Questo simbolo si chiama carattere di escape (ndt, pronunciato ischeip) e la sua funzione è di informare Python di stampare, in questo caso, il carattere apice singolo senza considerarlo come un delimitatore della stringa. Altri caratteri di escape (giusto per citarne qualcuno) sono '\n' che è l'a-capo e '\t' che rappresenta un tab. Li affronteremo più avanti con un esempio.

Assegnazione e uguaglianze

Dobbiamo imparare alcune altre cose prima di cominciare con il nostro prossimo esempio. La prima è la differenza tra l'assegnazione e l'uguaglianza. Abbiamo già usato l'assegnazione più volte nei nostri esempi. Quando vogliamo assegnare un valore ad una variabile, utilizziamo l'operatore di assegnazione '=' (il segno di uguale):

```
variabile  = valore
```

Quando invece vogliamo paragonare una variabile con un determinato valore, dobbiamo effettuare una comparazione. Diciamo di voler controllare se una variabile è uguale ad un determinato valore. In questo caso utilizzeremo il segno "==" (due segni di uguale):

```
variabile == valore
```

Perciò, se abbiamo una variabile che si chiama ciclo e vogliamo controllare se sia uguale, per esempio, al valore 12, ecco cosa faremo:

```
if ciclo == 12:
```

Per ora non preoccupatevi dell'if né dei due punti. L'unica cosa da ricordare è che per effettuare una comparazione dovremo usare il simbolo '=='.

Commenti

La prossima cosa che vedremo sono i commenti. I commenti sono importanti per diversi motivi. Non solo aiutano gli altri a capire quali fossero le nostre intenzioni nel codice, ma aiutano anche noi stessi a capire, passato un po' di tempo dalla scrittura, con quali finalità, ad esempio 6 mesi fa, abbiamo scritto quel determinato codice. Quando inizierete a scrivere diversi programmi, i commenti saranno molto importanti. I commenti servono anche per dire a Python di ignorare determinate righe di codice. Per commentare una riga si utilizza il segno '#'. Ad esempio:

```
# Questo è un commento
```

Potete inserire commenti in qualsiasi parte all'interno di una riga, ma tenete a mente che Python ignorerà tutto ciò che segue il segno '#'.

La dichiarazione if

Torniamo ora sulla dichiarazione "if" che abbiamo brevemente introdotto prima. Quando vogliamo effettuare una decisione in base al valore di qualcosa, possiamo utilizzare la dichiarazione if:

```
if ciclo == 12:
```

Questo comando controllerà la variabile 'ciclo' e, se il suo valore sarà 12, allora verrà eseguito tutto ciò che è contenuto nel blocco indentato successivo. Spesso ciò sarà sufficiente. Tuttavia, come potremmo fare se volessimo che "se una variabile è qualcosa allora fai questo, altrimenti fai quest'altro" ? In pseudo codice,

potremmo dire:

```
se x == y allora
  fai qualcosa
altrimenti
  fai qualcos'altro
```

In Python invece diremo:

```
if x == y:
  fai qualcosa
else:
  fai qualcos'altro
```

Le cose principali da ricordarsi sono:

1. terminate ogni dichiarazione if o else con il due punti

2. INDENTATE le righe del vostro codice

Se per caso aveste più di un valore da raffrontare, potreste usare il formato if/elif/else. Ad esempio

```
x = 5
if x == 1:
  print 'X vale 1'
elif x < 6:
  print 'X è minore di 6'
elif x < 10:
  print 'X è minore di 10'
else:
  print 'X è maggiore o uguale
a 10'
```

Notate che stiamo utilizzando l'operatore '<' per testare se x è MINORE DI un certo valore (in questo caso 6 o 10). Altri comuni operatori di paragone sono il maggiore di '>', minore o uguale a '<=', maggiore o uguale a '>=' e diverso da '!='.

La dichiarazione while

A questo punto, analizziamo la dichiarazione while. La dichiarazione while permette di creare dei cicli che saranno ripetuti fino a che una determinata soglia non verrà raggiunta (N.d.t. while == fintanto che). Un semplice esempio sarebbe quello di assegnare ad una variabile chiamata "ciclo" il valore 1. Quindi, fintanto che la variabile sia minore o uguale a 10, stampare il valore di 'ciclo' e aggiungere 1 a ciclo. Quando ciclo è superiore a 10, interrompere l'esecuzione.

```
ciclo = 1
while ciclo <= 10:
  print ciclo
  ciclo = ciclo + 1
```

Eseguito nel terminale, il risultato sarà il seguente:

```
1
2
3
4
5
6
7
8
9
10
```

Esattamente ciò che volevamo vedere. Nella Fig. 1 (in alto a destra) c'è un esempio simile ma, per quanto ancora semplice, leggermente più complesso.

In questo esempio abbiamo combinato la dichiarazione if, il ciclo while, la dichiarazione raw_input, l'operatore di assegnamento e l'operatore di confronto (tutti 8 righe di programma).

Se eseguirete questo programma, ecco cosa vedrete:

```
ciclo = 1
while ciclo == 1:
  risposta = raw_input("Scrivi qualcosa o scrivi 'stop' per
terminare => ")
  if risposta == 'stop':
    print 'sto terminando...'
    ciclo = 0
  else:
    print 'Hai scritto %s' % risposta
```

FIG. 1

```
Scrivi qualcosa o scrivi 'stop'
per terminare => RANA
Hai scritto RANA
Scrivi qualcosa o scrivi 'stop'
per terminare => uccello
Hai scritto uccello
Scrivi qualcosa o scrivi 'stop'
per terminare => 42
Hai scritto 42
Scrivi qualcosa o scrivi 'stop'
per terminare => STOP
Hai scritto STOP
Scrivi qualcosa o scrivi 'stop'
per terminare => stop
sto terminando...
```

Notate che quando ho scritto 'STOP' il programma non è terminato. Ciò è accaduto perché abbiamo raffrontato il valore della variabile 'risposta' con 'stop' (risposta == 'stop'). 'STOP' NON è uguale a 'stop'.

Un ulteriore veloce esempio prima di lasciarci per un altro mese. Diciamo di voler controllare se un utente ha il permesso per accedere al programma. Sebbene questo esempio non mostri il

modo migliore di effettuare un compito simile, è comunque utile per provare alcune delle cose che abbiamo imparato. In pratica, chiederemo all'utente di inserire nome e password, li confronteremo con le informazioni conservate nel nostro codice e decideremo, in base a ciò che troveremo, cosa permettere all'utente. Utilizzeremo due liste: una per contenere gli utenti autorizzati e una per contenere le password. Utilizzeremo poi raw_input per ottenere le informazioni da parte dell'utente e, infine, utilizzeremo la dichiarazione if/elif/else per controllare e decidere se l'utente è autorizzato. Ricordate: questo non è il metodo migliore per fare questo tipo di controlli. Analizzeremo un metodo più efficiente in un futuro articolo. Potete vedere il codice nel riquadro a destra.

```
#-----------------------------------------------
#password_test.py
# esempio su come utilizzare l'if/else, le liste, gli assegnamenti, raw_input
# i commenti e i confronti
#-----------------------------------------------
# Assegnazione dei nomi utente e delle password
utenti = ['Fred', 'John', 'Steve', 'Ann', 'Mary']
password = ['accesso', 'cane', '12345', 'neo', 'qwerty']
#-----------------------------------------------
# Richiedi nome utente e password
nomeutnt = raw_input('Inserisci il tuo nome utente => ')
pwd = raw_input('Inserisci la tua password => ')
#-----------------------------------------------
# Controlla che l'utente sia nella lista
if nomeutnt in utenti:
  posizione = utenti.index(nomeutnt) # Trova la posizione dell'utente nella lista degli utenti
  if pwd == password[posizione]:
    print 'Ciao %s. Hai il permesso per accedere.' % nomeutnt
  else:
    print 'La password inserita non e' valida. Accesso non consentito.'
else:
  print 'Spiacente... non ti conosco. Accesso non consentito.'
```

Salvate il codice in un file dal nome 'password_test.py' ed eseguitelo provando diverse combinazioni.

L'unica cosa di cui non abbiamo ancora parlato è la serie di controlli che iniziano con 'if nomeutnt is utenti:'. Ciò che stiamo facendo è un controllo per vedere se il nome che abbiamo inserito è contenuto nella lista. In caso positivo, dovremo trovare la posizione dell'utente nella lista. Utilizzeremo, quindi, utenti.index(nomeutnt) per sapere in che posizione della lista si trova l'utente e utilizzeremo questa informazione per estrarre la sua password conservata, nella stessa posizione, nell'altra lista. Per esempio 'John' è alla posizione 1 nella lista degli utenti. La sua password, 'cane', è nella posizione 1 della lista delle password. In questo modo potremo controllare le due informazioni. A questo punto, capire il programma dovrebbe essere abbastanza semplice.

Abbiamo visto abbastanza per questo mese. La prossima volta impareremo le funzioni e i moduli. Fino ad allora, prendete confidenza con quel che avete imparato e divertitevi.

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

Programmare in Python - Parte 3

Scritto da Greg Walters

VEDI ANCHE:
FCM nn. 27-28 - Python parti 1-2

VALIDO PER:
ubuntu kubuntu xubuntu

CATEGORIE:
Sviluppo Grafica Internet M/media Sistema

DISPOSITIVI:
CD/DVD HDD USB Drive Laptop Wireless

Nell'ultimo articolo, abbiamo studiato le liste, le sostituzioni letterali, i commenti, uguaglianza contro assegnamento, le istruzioni if e while. Vi avevo promesso che mi sarei occupato dei moduli e delle funzioni. Quindi iniziamo.

Moduli

I moduli sono una via per estendere la programmazione Python. È possibile crearne di propri, usare quelli a corredo di Python, o usare moduli creati da altri. Lo stesso Python è fornito di centinaia di moduli diversi che facilitano la programmazione. Un elenco dei moduli globali forniti con Python lo potete trovare su http://docs.python.org/modindex.html. Alcuni moduli sono specifici per un determinato sistema operativo, ma la maggior parte sono totalmente cross-platform (possono essere usati ugualmente su Linux, Mac e Microsoft Windows). Per usare un modulo esterno, bisogna importarlo nel programma. Uno dei moduli forniti con Python è 'random'. Tale modulo permette di generare numeri pseudo-casuali. Useremo il modulo come mostrato in alto a destra nel nostro primo esempio.

Esaminiamo ciascuna riga di codice. Le prime quattro sono commenti. Ne abbiamo parlato nell'articolo passato. La riga cinque dice a Python di usare il modulo random. Dobbiamo esplicitamente dire a Python di farlo.

```
#=======================================
# random_esempio.py
# Esempio dell'uso del modulo random
#=======================================
import random
# print 14 random integers
for cntr in range(1,15):
     print random.randint(1,10)
```

Nella riga sette viene usato 'for' per stampare 14 numeri casuali. La riga otto usa randint() per stampare un intero tra 1 e 10. Notare come sia necessario indicare a Python a quale modulo la funzione appartiene. Lo si fa (in questo caso) con random.randint. Perché creare moduli? Bene, se tutte le possibili funzioni fossero incluse in Python, non solo quest'ultimo diventerebbe enorme e lento, ma la correzione dei bug diventerebbe un incubo. Usando i moduli è possibile frammentare il codice in gruppi, ciascuno con un proprio scopo. Se, per esempio, non si ha necessità di funzioni database, non serve conoscere l'esistenza del modulo SQLite. Se però ne avrete bisogno è pronto per l'uso. (In realtà faremo uso dei moduli database più avanti in questa serie.)

Quando inizierete a programmare in Python, è probabile che creerete vostri moduli da riutilizzare successivamente, senza riscriverli. Se sarà necessario cambiare qualcosa in quel gruppo di codice, lo si potrà fare con meno probabilità di compromettere il programma principale. Ci sono dei limiti a tutto questo e ne parleremo successivamente. Ora, quando abbiamo usato precedentemente l'istruzione 'import random', abbiamo detto a Python di garantirci l'accesso a tutte le funzioni del modulo random. Se, al contrario, avessimo bisogno di accedere alla sola funzione randinit(), possiamo riscrivere l'istruzione import così:

```
from random import randint
```

Ora quando chiamiamo la nostra funzione non dobbiamo più usare l'identificatore 'random.'. Il nostro codice cambia così

```
from random import randint
# stampa 14 interi casuali
for cntr in range(1,15):
    print randint(1,10)
```

Funzioni

Quando abbiamo importato il modulo random, ne abbiamo usato la funzione randinit(). Una funzione è un blocco di codice creato per essere chiamato, anche più di una volta, che è più semplice da gestire e che ci risparmia la fatica di riscriverlo ripetutamente. In modo grossolano e indicativo, possiamo dire che, ogni volta che si ha la necessità di scrivere lo stesso codice più di una o due volte, quel codice è un buon candidato a diventare una funzione. Anche se i due esempi seguenti sono semplici, sono un buon punto di partenza per l'uso delle funzioni. Diciamo di prendere due numeri, sommarli, quindi moltiplicarli, e poi sottrarli, mostrando ogni volta valori e risultati. Per complicare le cose, dobbiamo ripetere tutto tre volte con tre serie di numeri. Il nostro esempio assomiglierà a quello mostrato a destra.

```
#semplice esempio
print 'Somma dei due numeri %d e %d = %d ' % (1,2,1+2)
print 'Moltiplicazione dei due numeri %d e %d = %d ' % (1,2,1*2)
print 'Sottrazione di due numeri %d e %d = %d ' % (1,2,1-2)
print '\n'
print 'Somma dei due numeri %d e %d = %d ' % (1,4,1+4)
print 'Moltiplicazione dei due numeri %d e %d = %d ' % (1,4,1*4)
print 'Sottrazione di due numeri %d e %d = %d ' % (1,4,1-4)
print '\n'
print 'Somma dei due numeri %d e %d = %d ' % (10,5,10+5)
print 'Moltiplicazione dei due numeri %d e %d = %d ' % (10,5,10*5)
print 'Sottrazione di due numeri %d e %d = %d ' % (10,5,10-5)
print '\n'
```

Non solo si tratta di tanto codice da digitare, ma può comportare errori, sia di battitura che in modifiche successive. Invece creeremo una funzione chiamata 'DoTwo' che prende due numeri, compie i calcoli e stampa ogni volta l'output. Inizieremo usando la parola chiave 'def' (che avvisa che ci apprestiamo a definire una funzione). Dopo 'def' immettere mo il nome scelto per la funzione e quindi un elenco di parametri (se ce ne sono) tra parentesi. Questa riga è terminata da due punti (:). Il codice nella funzione è indentato. Il nostro esempio migliorato (#2) è mostrato in basso.

Come potete vedere, c'è meno codice da inserire - 8 righe invece di 12. Se dobbiamo cambiare qualcosa nella nostra funzione, è possibile farlo senza causare troppi problemi al programma principale. Chiamiamo la nostra funzione, in questo caso, usando il suo nome seguito dai parametri.

Ecco un altro esempio di funzione. Consideriamo i seguenti requisiti.

Vogliamo creare un programma che stampi in

```
#semplice esempio 2 ancora semplice, ma migliore
def DoTwo(num1,num2):
    print 'Somma di due numeri %d e %d = %d ' % (num1,num2,num1+num2)
    print 'Moltiplicazione di due numeri %d e %d = %d ' % (num1,num2,num1*num2)
    print 'Sottrazione di due numeri %d e %d = %d ' % (num1,num2,num1-num2)
    print '\n'
DoTwo(1,2)
DoTwo(1,4)
DoTwo(10,5)
```

maniera elegante un elenco di oggetti acquistati. Deve assomigliare al testo sotto.

Il costo di ciascun oggetto e il totale di tutti gli stessi sarà formattato come dollari e centesimi. La larghezza dell'output dovrà essere variabile. I valori a destra e sinistra saranno anch'essi variabili. Useremo 3 funzioni per raggiungere lo scopo. Una stamperà la prima e ultima riga, una stamperà le righe con i dettagli degli oggetti inclusa la riga con il totale e una stamperà la riga separatore. Fortunatamente Python ci fornisce una serie di funzionalità che facilitano il compito. Ricorderete che moltiplicammo una stringa per 4 e che ci vennero restituite quattro copie della stessa stringa. Possiamo usare quella tecnica anche in questo caso. Per stampare le righe in alto e in basso prendiamo la larghezza desiderata, sottraiamo due per i due caratteri + utilizzando la formula " '=' * (larghezza-2)". Per rendere le cose ancora più semplici, useremo la sostituzione di variabile per mantenere tutti gli oggetti su una riga. La stringa da stampare sarà così codificata ('+',('=' * larghezza-2)),'+'). Potremmo ora far sì che la nostra routine stampi la stringa direttamente, ma useremo la parola chiave return per inviare la stringa generata alla riga chiamante. Chiameremo questa funzione 'SopraOrSotto' e il codice della funzione sarà questo:

```
def SopraOrSotto(larghezza):
    # 'larghezza' è la
larghezza totale della riga
che verrà restituita
    return '%s%s%s' %
('+',('=' * (larghezza-
2)),'+')
```

```
'+==============================+'
'| Oggetto 1                X.XX |'
'| Oggetto 2                X.XX |'
'|------------------------------|'
'| Totale                   X.XX |'
'+==============================+'
```

Potremmo tralasciare il commento, ma è utile sapere a colpo d'occhio il significato del parametro 'larghezza'. Per chiamarla, possiamo scrivere 'print SopraOrSotto(40)' o qualunque altra larghezza desideriamo per la riga. Ora abbiamo una funzione che si occupa di due delle righe. Passiamo ora a creare una nuova funzione che si occupi della riga separatore usando un codice simile... oppure possiamo modificare la funzione appena scritta per includere il parametro del carattere da usare in mezzo ai segni +. Facciamolo. Possiamo continuare a chiamarla SopraOrSotto.

```
def
SopraOrSotto(carattere,larghe
zza):
    # 'larghezza' è la
larghezza totale della riga
ritornata
    # 'carattere' è il
carattere da inserire tra i
'+'
    return '%s%s%s' %
('+',(carattere * (larghezza-
2)),'+')
```

Ora è possibile constatare l'utilità dei commenti. Ricordate, faremo restituire la stringa generata, così dobbiamo avere qualcosa da ricevere quando la chiamiamo. Invece di assegnarla ad un'altra stringa, semplicemente la stamperemo. Ecco la riga chiamante.

```
print SopraOrSotto('=',40)
```

Così adesso non solo ci siamo occupati di tre righe, ma abbiamo ridotto il numero delle routine da 3 a 2. Così non ci resta che stampare la parte centrale. Chiamiamo la prossima funzione 'Fmt'. Le passeremo 4 parametri come segue:

val1 - il valore da stampare a sinistra
col_sn - la larghezza di questa "colonna"
val2 - il valore da stampare a destra (che dovrebbe essere un valore con virgola)
col_dx - la larghezza di questa "colonna"

Il primo compito è formattare l'informazione della parte destra. Poiché vogliamo rappresentare il valore in dollari e centesimi, si può usare una funzione speciale per la sostituzione di variabile che dice: stampa il valore come numero con virgola con n decimali. Il comando sarà '%2.f'. Assegnerò questo valore alla variabile 'part2'. Così la nostra riga sarà 'part2 = '%.2f'

% val2'. Possiamo usare anche una serie di funzioni incluse nel modulo string di Python chiamate ljust e rjust. ljust giustificherà la stringa a sinistra, riempiendo la parte destra con qualsiasi carattere si voglia. rjust fa la stessa cosa, eccetto che il riempimento va a sinistra. Ora la parte interessante. Usando le sostituzioni metteremo insieme una stringa bella corposa e la ritorneremo al codice chiamante. Ecco la nostra prossima riga.

```
return 'ss' % ('| 
',val1.ljust(col_sn-
2,''),part2.rjust(col_dx-
2,''),' |')
```

Non lasciatevi scoraggiare dall'aspetto, dissezioniamola per vedere quanto in realtà sia semplice:
return - Restituiremo la nostra stringa al codice chiamante.
'ss' - Andremo a racchiudere quattro valori nella stringa. Ciascun %s è un segna posto.
(- Inizia la lista variabile
'| ' - Stampa questi caratteri
val1.ljust(col_sn-2),' ') - Prende la variabile val1 che abbiamo passato, la giustifichiamo a sinistra con spazi per (col_sn-2) caratteri. Sottraiamo per permettere '| ' sul lato sinistro.
part2.rjust(col_dx-2),' ') - Giustifichiamo a destra la stringa al prezzo di rightbit-2 spazi. ' |' termina la stringa.

Questo è quanto. Anche se potevo inserire un sistema per il controllo degli errori, lascio questo compito a voi. Quindi... la nostra funzione Fmt è di sole due righe senza la definizione e i commenti. La possiamo chiamare in questo modo.

```
print
Fmt('Oggetto
1',30,oggetto1,10)
```

Di nuovo, avremmo potuto assegnare il valore di ritorno ad un'altra stringa, ma semplicemente la stampiamo. Notate che abbiamo inviato 30 per la larghezza della parte sinistra e 10 per quella destra. Questi uguagliano il 40 inviato precedentemente alla routine SopraOrSotto. Allora, avvia il tuo editor e inserisci il codice sottostante.

```
+======================================+
| Oggetto 1                      3.00 |
| Oggetto 2                     15.00 |
+--------------------------------------+
| Totale                        18.00 |
+======================================+
```

Salvate il codice come

```
#pprint1.py
#Esempio di funzioni quasi-utili

def SopraOrSotto(carattere,larghezza):
    # 'larghezza' è la larghezza totale della riga ritornata
    return '%s%s%s' % ('+',(carattere * (larghezza-2)),'+')

def Fmt(val1,col_sn,val2,col_dx):
    # stampa due valori riempiti con spazi
    # val1 è stampato a sinistra, val2 è stampato a destra
    # col_sn è la larghezza della parte sinistra, col_sn è la larghezza della parte destra
    part2 = '%.2f' % val2
    return '%s%s%s%s' % ('| ',val1.ljust(col_sn-2,' '),part2.rjust(col_dx-2,' '),' |')
# Definiamo il prezzo di ciascun oggetto
oggetto1 = 3.00
oggetto2 = 15.00
# Ora stampiamo il tutto...
print SopraOrSotto('=',40)
print Fmt('Oggetto 1',30,oggetto1,10)
print Fmt('Oggetto 2',30,oggetto2,10)
print SopraOrSotto('-',40)
print Fmt('Totale',30,oggetto1+oggetto2,10)
print SopraOrSotto('=',40)
```

'pprint1.py' ed eseguitelo. L'output dovrebbe assomigliare al testo in alto a destra della pagina precedente.

Anche se questo è un esempio molto semplice, vi dovrebbe dare una buona idea su come usare le funzioni. Ora, estendiamo ancora un pò il concetto e impariamo qualcosa sulle liste. Ricordate quando nella parte 2 abbiamo iniziato a discuterne? Bene, una cosa che ho tralasciato è che una lista può contenere qualunque cosa, comprese altre liste. Dichiariamo una nuova lista nel nostro programma chiamata oggetti e riempiamola così:

```
oggetti =
[['Soda',1.45],['Caramelle',.75
],['Pane',1.95],['Latte',2.59]]
```

Se volessimo accedervi come fosse una normale lista useremmo print oggetti[0]. Però, ciò che otterremmo è ['Soda',1.45], che non è proprio quello che cercavamo. Noi vogliamo accedere a ciascun oggetto della lista. Così useremo 'print oggetti[0][0]' per 'Soda' e [0][1] per recuperarne il costo di 1.45. Ora, abbiamo 4 oggetti che sono stati acquistati e vogliamo usare questa informazione nella nostra routine print. L'unica modifica da apportare riguarda la parte finale del programma. Salviamo l'ultimo programma come 'pprint2.py', quindi commentate la definizione dei due oggettox e inseriamo la lista di cui sopra. Ora dovrebbe assomigliare a questo.

```
#oggetto1 = 3.00
#oggetto2 = 15.00
oggetti
=[['Soda',1.45],['Caramelle',.7
5],['Pane',1.95],['Latte',2.59]
]
```

Dopo, rimuovete tutte le righe che chiamano Fmt(). Quindi aggiungete le seguenti righe (con #NUOVA RIGA alla fine) per rendere il vostro codice come quello mostrato a destra.

```
oggetti = [['Soda',1.45],['Caramelle',.75],['Pane',1.95],['Latte',2.59]]

print SopraOrSotto('=',40)

totale = 0 #NUOVA RIGA
for cntr in range(0,4): #NUOVA RIGA
    print Fmt(oggetti[cntr][0],30,oggetti[cntr][1],10) #NUOVA RIGA
    totale += oggetti[cntr][1] #NUOVA RIGA
print SopraOrSotto('-',40)
print Fmt('Totale',30,totale,10) #RIGA CAMBIATA
print SopraOrSotto('=',40)
```

Ho impostato una variabile contatore nel for che cicla nella lista per ciascun suo elemento. Notate che ho anche impostato una variabile chiamata totale. Abbiamo impostato totale a 0 prima di entrare nel for. Quindi quando si stampa ciascun oggetto venduto, aggiungiamo il costo al totale. Infine, stampiamo il totale subito dopo la riga separatore. Salvate il programma ed eseguitelo. Dovreste ottenere qualcosa di simile al testo in basso.

Se siete un po' folli, potete aggiungere una riga per le tasse. Fatelo imitando la riga totale, ma usando (totale * .086) come costo.

```
print
Fmt('Tasse:',30,totale*.086,10)
```

Se volete, potete aggiungere altri oggetti alla lista e vedere come va. Questo è tutto per questa parte. La prossima volta ci concentreremo sulle classi. **Buon divertimento!**

```
+======================================+
| Soda                           1.45  |
| Caramella                      0.75  |
| Pane                           1.95  |
| Latte                          2.59  |
+--------------------------------------+
| Totale                         6.74  |
+======================================+
```

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

Programmare in Python - Parte 4

VEDI ANCHE:
FCM#27-29 - Python Parti 1-3

ubuntu kubuntu xubuntu

CATEGORIE:

DISPOSITIVI:


```
class Dog():
    def __init__(self,dogname,dogcolor,dogheight,dogbuild,dogmood,dogage):
        #here we setup the attributes of our dog
        self.name = dogname
        self.color = dogcolor
        self.height = dogheight
        self.build = dogbuild
        self.mood = dogmood
        self.age = dogage
        self.Hungry = False
        self.Tired = False
```

Vi avevo promesso, la scorsa volta, che avremmo parlato delle classi. Così, ecco su cosa ci concentreremo. Cosa sono le classi e a che cosa servono?

La classe è un mezzo per la costruzione di oggetti. Un oggetto è semplicemente un modo di raggruppare attributi e comportamenti. So che sembra confuso, ma continuiamo. Ragionate in questo modo. Un oggetto è un modello di qualcosa del mondo reale. La classe è una via per implementarlo. Per esempio, a casa ho tre cani. Un Beagle, un Labrador e un Pastore Tedesco/Blue Heeler. Tutti sono cani, ma sono anche differenti tra loro. Accanto ad attributi comuni, troviamo anche differenze. Per esempio, il Beagle è piccolo, grassottello, marrone e irritabile. Il Labrador è di taglia media, nero e tranquillo. Il Pastore è alto, magro, nero e un pò pazzerello. Ovviamente, alcuni attributi sono ovvi. Piccolo/taglia media/alto sono tutti attributi collegati all'altezza. Irritabile/tranquillo/pazzerello sono attributi dell'umore. Dal punto dei vista dei comportamenti, possiamo considerare mangiare, dormire, giocare, e altre azioni.

Tutti e tre saranno della classe 'Cane'. Considerando gli attributi usati per descriverli, potremmo usare Cane.Nome, Cane.Altezza, Cane.Corporatura (magro,grassottello, ecc.) e Cane.Colore. Poi abbiamo i comportamenti come Cane.Abbaia, Cane.Mangia,Cane.Dormi, e così via.

Come detto prima, i tre cani sono di razze distinte. Ciascuna razza sarà una sottoclasse della classe Cane. In un diagramma, sarebbero così rappresentati.

```
          /--Beagle
Dog ---|-- Lab
          \--Shepherd/Heeler
```

Ciascuna sottoclasse eredita gli attributi della classe Cane. Quindi, se creiamo una istanza di Beagle, essa erediterà tutti gli attributi dalla sua classe genitore, Cane.

```
Beagle = Dog()
Beagle.Name = 'Archie'
Beagle.Height = 'Short'
Beagle.Build = 'Chubby'
Beagle.Color = 'Brown'
```

Inizia ad avere senso?

Allora, creiamo la nostra classe Cane (mostrata sopra). Inizieremo con la parola chiave "class" e il nome della classe.

Prima di procedere ulteriormente, osserviamo la funzione che abbiamo appena definito. La funzione init (due underscore + 'int' + due underscore) è una funzione di inizializzazione utilizzata da ogni classe. Questa routine è la prima ad essere eseguita quando chiamiamo la classe nel nostro codice. In questo caso abbiamo usato una serie di parametri per impostarne alcune semplici informazioni; abbiamo le variabili nome, colore, altezza, corporatura, umore, età, affamato e stanco. Ci ritorneremo su a breve. Ora aggiungiamo altro codice.

```
Beagle =
Dog('Archie','Brown','Short',
'Chubby','Grumpy',12)
print Beagle.name
print Beagle.color
print Beagle.mood
print Beagle.Hungry
```

Si tratta di codice NON INDENTATO che risiede al di fuori della classe. Con la prima riga si crea una istanza della classe cane chiamata Beagle. Nel farlo, abbiamo passato delle informazioni come il nome, il colore, ecc, del Beagle. Le quattro righe successive servono a recuperare le informazioni. Ancora altro codice. Aggiungete alla classe le righe del riquadro in alto a destra, dopo la funzione init.

Le possiamo richiamare con Beagle.Mangia() o Beagle.Dormi(). Aggiungiamo un altro metodo. Lo chiameremo Abbaia. Il suo codice è mostrato a destra.

Questa funzione l'ho resa più flessibile. A seconda dell'umore del cane, il guaito cambierà. Nella prossima pagina c'è l'intero codice della classe.

Così, eseguendolo otterremo:

```
My name is Archie
My color is Brown
My mood is Grumpy
I am hungry = False
Sniff Sniff...Not Hungry
Yum Yum...Num Num
GRRRRR...Woof Woof
```

```
    def Eat(self):
        if self.Hungry:
            print 'Yum Yum...Num Num'
            self.Hungry = False
        else:
            print 'Sniff Sniff...Not Hungry'
    def Sleep(self):
        print 'ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ'
        self.Tired = False
```

```
    def Bark(self):
        if self.mood == 'Grumpy':
            print 'GRRRRR...Woof Woof'
        elif self.mood == 'Laid Back':
            print 'Yawn...ok...Woof'
        elif self.mood == 'Crazy':
            print 'Bark Bark Bark Bark Bark Bark Bark'
        else:
            print 'Woof Woof'
```

Questo è quanto per il vecchio irritabile Beagle. Però, prima, avevo detto di avere 3 cani. Avendo codificato attentamente la classe, quello che ci resta da fare è crearne altre due istanze.

```
Lab =
Dog('Nina','Black','Medium','
Heavy','Laid Back',7)
Heeler =
Dog('Bear','Black','Tall','Sk
inny','Crazy',9)
print 'My Name is %s' %
Lab.name
print 'My color is %s' %
Lab.color
print 'My Mood is %s' %
Lab.mood
print 'I am hungry = %s' %
Lab.Hungry
Lab.Bark()
Heeler.Bark()
```

Notate come abbia creato le istanze di entrambi i cani prima delle istruzioni print. Non è un problema, perché ho "definito" le istanze prima di aver chiamato uno dei metodi. Ecco l'output di tutto il programma:

```
My name is Archie
My color is Brown
My mood is Grumpy
I am hungry = False
Sniff Sniff...Not Hungry
```

```
Yum Yum...Num Num
GRRRRR...Woof Woof
My Name is Nina
My color is Black
My Mood is Laid Back
I am hungry = False
Yawn...ok...Woof
Bark Bark Bark Bark Bark
Bark Bark
```

Ora che avete le basi, il vostro compito sarà espandere la classe per includere nuovi metodi, come Gioca o IncontraCaneStrano o simili.

La prossima volta discuteremo della programmazione di GUI o Interfaccia Grafica Utente. Useremo per lo scopo Boa Constructor.

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.

```
class Dog():
    def __init__(self,dogname,dogcolor,dogheight,dogbuild,dogmood,dogage):
        #here we setup the attributes of our dog
        self.name = dogname
        self.color = dogcolor
        self.height = dogheight
        self.build = dogbuild
        self.mood = dogmood
        self.age = dogage
        self.Hungry = False
        self.Tired = False

    def Eat(self):
        if self.Hungry:
            print 'Yum Yum...Num Num'
            self.Hungry = False
        else:
            print 'Sniff Sniff...Not Hungry'

    def Sleep(self):
        print 'ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ'
        self.Tired = False

    def Bark(self):
        if self.mood == 'Grumpy':
            print 'GRRRRR...Woof Woof'
        elif self.mood == 'Laid Back':
            print 'Yawn...ok...Woof'
        elif self.mood == 'Crazy':
            print 'Bark Bark Bark Bark Bark Bark Bark'
        else:
            print 'Woof Woof'

Beagle = Dog('Archie','Brown','Short','Chubby','Grumpy',12)
print 'My name is %s' % Beagle.name
print 'My color is %s' % Beagle.color
print 'My mood is %s' % Beagle.mood
print 'I am hungry = %s' % Beagle.Hungry
Beagle.Eat()
Beagle.Hungry = True
Beagle.Eat()
Beagle.Bark()
```

Programmare in Python - Parte 5

Scritto da Greg Walters

VEDI ANCHE:
FCM nn. 27-30 - Python parti 1-4

VALIDO PER:
ubuntu kubuntu xubuntu

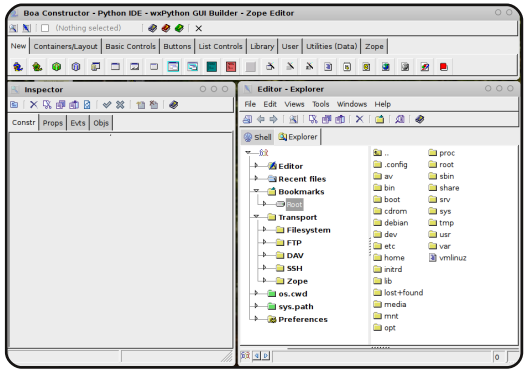
CATEGORIE:
Sviluppo Grafica Internet M/media Sistema

DISPOSITIVI:
CD/DVD HDD USB Drive Laptop Wireless

Se siete come me, ODIERETE la prima parte di questa lezione. ODIO quando l'autore mi dice di leggere due volte ogni parola nel suo libro/capitolo/articolo, perché SO già che sarà una noia - anche quando so che è per il mio bene, e finirò col farlo comunque.

Consideratevi avvertiti. PER FAVORE leggete i successivi noiosi paragrafi con attenzione. Presto arriveremo alla parte divertente, ma abbiamo bisogno di preparare il terreno prima di iniziare a programmare.

INNANZITUTTO dovete installare Boa Constructor e wxPython. Usate Synaptic e selezionate sia wxPython che Boa Constructor. Una volta installati, dovreste trovare Boa in Applicazioni\Programmazione\Boa Contructor. Procedete ed avviatelo. Questo renderà le cose un po' più semplici. Una volta avviato il programma, vedrete tre distinte finestre: una in alto e due in basso. Potrebbe essere necessario ridimensionarle e spostarle un po', fino a sistemarle in questa maniera:



La finestra superiore è quella strumenti. Quella in basso a sinistra è l'inspector, e quella in basso a destra è l'editor. Nella finestra strumenti, troverete varie linguette (Nuovo, Contenitore/Posizionamento, etc) che vi permetteranno di iniziare un nuovo progetto, aggiungere frame ad un progetto esistente, e aggiungere vari controlli all'applicazione. La finestra inspector acquisterà importanza quando inizieremo ad aggiungere controlli. La finestra editor ci consente di modificare il codice, salvare i progetti, e molto altro. Ritorniamo a quella strumenti, ed osserviamo ciascuna linguetta - partendo da quella "Nuovo". Anche se qui ci sono molte opzioni, ne tratteremo solo due. Sono il 5° e il 6° pulsante da sinistra: wx.App e wx.Frame. wx.App ci permette di creare un'applicazione completa partendo da due file auto-generati. Uno per il frame ed un altro per l'applicazione. Questo è il metodo che preferisco usare. wx.Frame è usato per aggiungere altri frame alla nostra applicazione e/o creare una applicazione standalone da un singolo file sorgente. Ne parleremo più tardi.

Osserviamo ora la linguetta Contenitore/Posizionamento. Ci sono parecchie cose. Quelle che userete maggiormente sono wx.panel (prima da sinistra) e i ridimensionatori (2,3,4,5 e 6 da destra). Sotto Controlli base, troverete etichette, box di testo, caselle, pulsanti radio, e altri. Sotto Bottoni, troverete vari tipi di pulsanti. Controlli lista comprende griglie per dati ed altre liste. In Utilità troveremo timer e oggetti per menu.

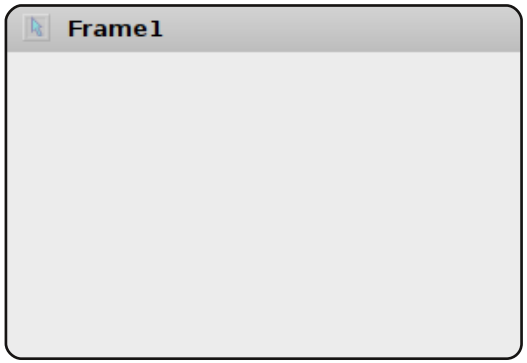
Ecco alcune cose da ricordare quando siamo pronti per la prima applicazione. Ci sono alcuni bug nella versione Linux. Uno riguarda l'impossibilità di muovere ALCUNI controlli nel designer. Usate la combinazione <Ctrl>+Tasti Freccia per muoverli o aggiustarne la posizione. Un altro riguarda i tutorial allegati a Boa

Constructor - posizionando un pannello è difficile seguirli. Osservate i piccoli riquadri (ve li spiegherò presto). Potete anche usare la linguetta Objs dell'inspector e in questo modo selezionarlo.

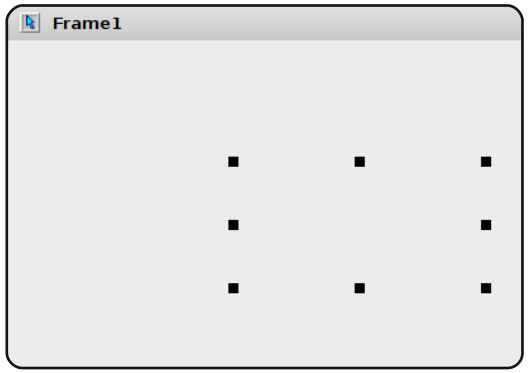
Ok, continuiamo. Nella linguetta 'Nuovo' della finestra strumenti, selezioniamo wx.App (5° bottone da sinistra). Verranno così create due nuove linguette nell'editor: una chiamata “*(App1)*”, l'altra “*(Frame1)*”. Che ci crediate o meno, la PRIMA cosa da fare è salvare i nostri due file, partendo da Frame1. Il pulsante per salvare è il 5° da sinistra nell'Editor. Si aprirà la finestra "Salva come" chiedendoci dove salvare e come chiamare il file. Create nella vostra home una cartella chiamata GuiTest, e salvate il file come “Frame1.py”. Osservate che la linguetta “*(Frame1)*” ora mostra “Frame1”. (I caratteri “*(“ ci informano che il file deve essere salvato.) Ora fate lo stesso con la linguetta App1.

Ora esaminiamo alcuni dei pulsanti della barra degli strumenti di Editor. Quelli per ora interessanti sono Salva (5° da sinistra) ed Esegui (freccia gialla, 7° da sinistra). Se vi trovate in una linguetta frame (Frame1 per esempio) ci saranno pulsanti extra che avrete bisogno di conoscere. Partiamo dal pulsante Designer:

È importante. Permette di disegnare la finestra della nostra GUI - che è quello che faremo. Quando ci cliccherete vi sarà mostrata una finestra vuota.



Si tratta di una tela bianca in cui inserire i controlli necessari (secondo logica). La prima cosa che vogliamo fare è posizionare un wx.panel. Quasi tutto quello che ho letto dice di non mettere controlli (ad eccezione di wx.panel) direttamente su un frame. Quindi, fate clic sulla linguetta Contenitore/Posizionamento nella finestra Strumenti, quindi sul bottone wx.Panel. Quindi, passare al frame sul quale si sta lavorando e cliccare in un punto qualunque all'interno del frame. Saprete che ha funzionato se vedete qualcosa simile a questo:

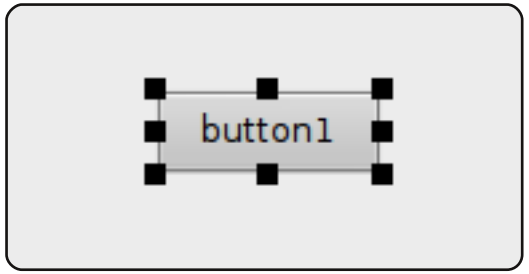


Ricordate quando vi ho avvisato dei bug? Bene, questo è un altro. Non preoccupatevi. Vedete gli 8 piccoli quadratini neri? Sono i limiti del pannello. Se volete potete cliccare e trascinarne uno per ridimensionare il pannello, ma per il nostro progetto vogliamo che copra l'intero frame. A questo punto ridimensionate un poco solo il FRAME. Ora abbiamo un pannello su cui inserire altri controlli. Muovere il frame su cui si sta lavorando fino a che la barra strumenti dell'Editor non sia visibile. Sono comparsi due nuovi pulsanti: un "check" e una "X". La "X" annulla i cambiamenti apportati.

Il pulsante Check:

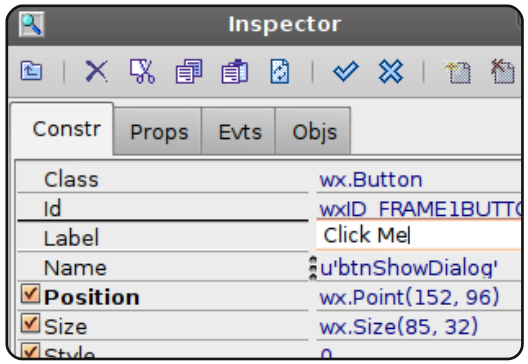
è chiamato pulsante "Invia". Serve per scrivere i cambiamenti sul file frame. Abbiamo ancora bisogno di salvarlo, ma questo includerà le novità nel file. Quindi, fate clic sul pulsante Invia. C'è un altro Invia nell’inspector, ma ce ne occuperemo dopo. Ora salvate il vostro file.

Ritornate alla modalità Designer. Cliccare sulla linguetta Bottoni della finestra Strumenti e quindi fate clic sul primo pulsante a sinistra, wx.Button. Quindi aggiungetelo al centro del frame. Avrete qualcosa di simile a questo:



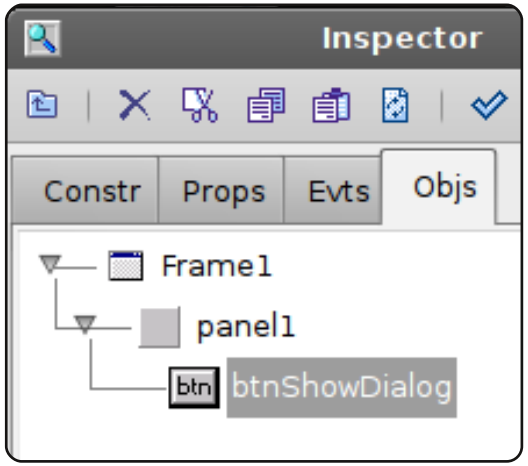
Notate come vi siano intorno 8 piccoli quadrati neri come per il pannello. Sono maniglie per ridimensionare. Mostrano inoltre quale controllo è selezionato. Per posizionarlo con cura al centro

del frame, tenete premuto il tasto Ctrl e usate i tasti freccia per muoverlo dove si vuole. Ora osservate l'inspector. Ci sono quattro linguette. Cliccare su 'Constr'. Da qui possiamo cambiare etichetta, nome, posizione, dimensione e stile. Per ora cambiamo il nome in 'btnShowDialog' e la proprietà Label in 'Cliccami'.



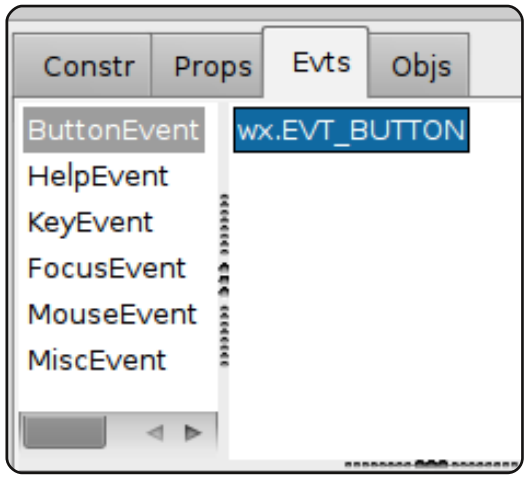
Ora, trascuriamo il resto della linguetta e passiamo a quella Objs. Questa mostra tutti i controlli usati e la loro relazione genitore/figlio. Come potete vedere, il pulsante è figlio di panel1, che è figlio di Frame1.

Premete Invia e salvate i cambiamenti. Ritornate ancora una volta al designer, e notate che (assumendo che sia ancora selezionata la linguetta Objs), Frame1 è ora selezionato. Questo va bene perché è quello



che ci serve. Ritornare alla linguetta 'Constr', e cambiare il titolo da 'Frame1' a 'La nostra prima GUI'. Cliccate Invia e salvate ancora una volta. Ora eseguiamo la nostra applicazione. Cliccate il pulsante giallo Esegui nell'Editor.

Cliccate quanto volete sul pulsante, ma non accadrà nulla. Perché? Bene, non abbiamo detto al pulsante cosa fare. Per questo, dobbiamo impostare un evento che deve verificarsi quando l'utente clicca il nostro bottone. Cliccare sulla X nell'angolo in alto a sinistra per interrompere l'esecuzione del frame. Poi, ripassare al designer, selezionate il pulsante e andate nella linguetta 'Evts' dell'Ispezionatore. Cliccate su ButtonEvent e quindi doppio click su wx.EVT_BUTTON, e notate che nella finestra sotto otteniamo un pulsante evento chiamato 'OnBtnShowDialogButton'. Clic su Invia e salvate.



Prima di andare oltre, guardiamo cosa abbiamo ottenuto sotto forma di codice (pagina 11).

La prima riga è un commento che dice a Boa Constructor che si tratta di un file boa. È ignorato dal compilatore Python, ma non da Boa. La riga successiva importa wxPython. Ora saltiamo alla definizione della classe.

All'inizio, c'è il metodo init_ctrls. Notate il commento proprio sotto la definizione. Non modificate il codice di questa sezione. Se lo fate, lo rimpiangerete. Ovunque SOTTO questa routine dovrebbe essere sicuro. In questa routine, troverete le definizioni di ciascun controllo del nostro frame.

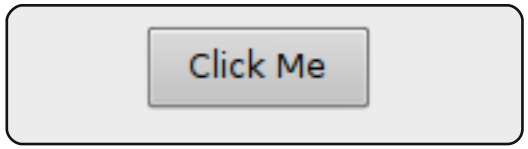
Quindi, osservate la routine __init__. Qui potete inserire qualunque chiamata al codice di inizializzazione. Infine, la routine OnBtnShowDialogButton. Qui è dove metteremo il codice da eseguire quando l'utente clicca il pulsante. Osservate che ora qui c'è la riga event.Skip(). In parole semplici, dice di uscire dalla routine quando si richiama l'evento.

Ora, quello che faremo è chiamare un finestra messaggio da far comparire con del testo. Si tratta di un'operazione comune per i programmatori per portare a conoscenza l'utente di qualcosa - un errore, o che un processo è terminato. In questo caso, chiameremo l'inclusa routine wx.MessageBox. La routine viene chiamata con due parametri. Il primo è il testo che

desideriamo mostrare nel messaggio e il secondo è il titolo. Commentate la riga event.Skip() e inserite la seguente riga.

```
wx.MessageBox('Hai cliccato il pulsante', 'Info')
```

Salvate e fate clic sul pulsante Esegui (freccia gialla). Dovreste vedere qualcosa simile



a questo:

E quando fate clic sul

pulsante dovreste vedere qualcosa di simile a questo:

Beninteso, questo è il modo più semplice per chiamare un messagebox. È possibile usare ulteriori parametri.

Ecco un breve sommario su come cambiare il tipo di icona nel box messaggio (di più la prossima volta).

wxICON_QUESTION - Mostra un'icona di domanda

wxICON_EXCLAMATION - Mostra un'icona di avviso

wxICON_ERROR - Mostra un'icona di errore

wxICON_INFORMATION - Mostra un'icona di informazione

La maniera di usarle sarebbe

```
wx.MessageBox('Hai cliccato il pulsante, 'Info', wx.ICON_INFORMATION)
```

```
#Boa:Frame:Frame1
import wx
def create(parent):
    return Frame1(parent)
[wxID_FRAME1, wxID_FRAME1BTNSHOWDIALOG, wxID_FRAME1PANEL1,
] = [wx.NewId() for _init_ctrls in range(3)]

class Frame1(wx.Frame):
    def _init_ctrls(self, prnt):
        # generated method, don't edit
        wx.Frame.__init__(self, id=wxID_FRAME1, name='', parent=prnt,
              pos=wx.Point(543, 330), size=wx.Size(458, 253),
              style=wx.DEFAULT_FRAME_STYLE, title=u'Our First GUI')
        self.SetClientSize(wx.Size(458, 253))
        self.panel1 = wx.Panel(id=wxID_FRAME1PANEL1, name='panel1', parent=self,
              pos=wx.Point(0, 0), size=wx.Size(458, 253),
              style=wx.TAB_TRAVERSAL)
        self.btnShowDialog = wx.Button(id=wxID_FRAME1BTNSHOWDIALOG,
              label=u'Click Me', name=u'btnShowDialog', parent=self.panel1,
              pos=wx.Point(185, 99), size=wx.Size(85, 32), style=0)
        self.btnShowDialog.Bind(wx.EVT_BUTTON, self.OnBtnShowDialogButton,
              id=wxID_FRAME1BTNSHOWDIALOG)

    def __init__(self, parent):
        self._init_ctrls(parent)
    def OnBtnShowDialogButton(self, event):
        event.Skip()
```

o qualunque icona si voglia usare in base alla situazione. Ci sono altre opzioni per i pulsanti di cui parleremo la prossima volta.

Così, fino ad allora, provate qualcuno degli altri controlli, disposizioni, e così via. Buon divertimento!

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

HOW-TO

Scritto da Greg Walters

Programmare in Python - Parte 6

VEDI ANCHE:
FCM nn. 27-31 - Python Parti 1-5

VALIDO PER:
ubuntu kubuntu xubuntu

CATEGORIE:
Sviluppo Grafica Internet M/media Sistema

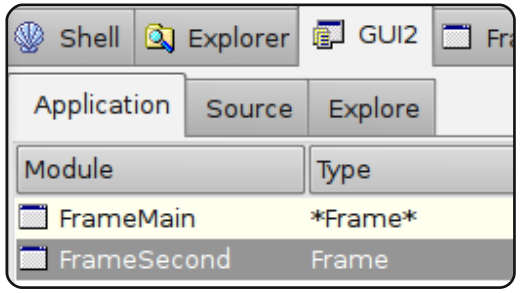
DISPOSITIVI:
CD/DVD HDD USB Drive Laptop Wireless

Spero che abbiate sperimentato Boa Constructor dal nostro ultimo incontro. Cominceremo con un programma molto semplice che mostrerà un frame, quindi aggiungeremo un pulsante per mostrarne un secondo. L'altra volta creammo una finestra di avviso. Questa volta creeremo un frame completamente a parte. Questa procedura è utile per i programmi con molti frame o finestre. Quindi... cominciamo...

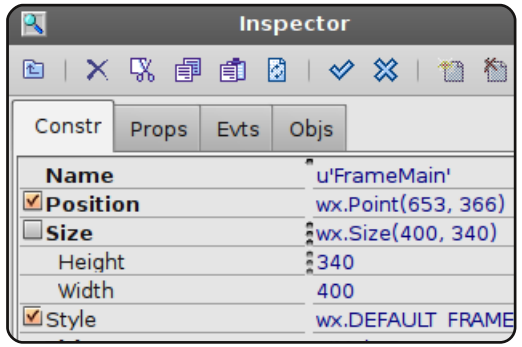
Avviate Boa Constructor e chiudete tutte le linguette della finestra Editor ad eccezione di Shell ed Esploratore usando la combinazione Ctrl-W. Questo ci assicurerà di partire da zero. Ora create un nuovo progetto facendo clic sul pulsante wx.App (rivedete il precedente articolo se necessario).

Prima di procedere oltre, salvate Frame1 come "FrameMain.py" e App1 come "Gui2.py". Passaggio importante. Con la linguetta GUI2 selezionata nella finestra Editor, ritornare alla linguetta Nuovo della finestra Strumenti e aggiungere un altro frame al nostro progetto facendo clic su wx.Frame (che si trova a fianco di wx.App). Assicurarsi che la linguetta Applicazioni mostri entrambi i frame sotto la colonna Modulo. Ora ritornate al nuovo frame e salvatelo come "FrameSecond.py":

Ora aprite FrameMain nel designer. Aggiungete un wx.Panel. Ridimensionatelo fino



a che il pannello copra il frame. Quindi cambiamo alcune proprietà: non l'abbiamo fatto l'ultima volta. Nella finestra Ispezionatore, assicurarsi che la linguetta Constr sia selezionata e impostare Title a "Main Frame" e Name a "FrameMain". Parleremo in seguito delle convenzioni per i nomi. Impostate la dimensione a 400x340 facendo clic sul check box Size. Verranno così mostrati high (altezza) e width (larghezza). L'altezza dovrebbe essere 400 e la larghezza 340:

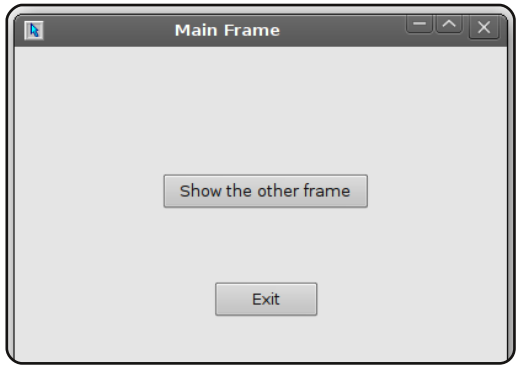


Ora fate clic sulla linguetta Props. Fate clic sulla proprietà Centered e settatela a wx.BOTH. Fate clic sul pulsante Invia e salvate il vostro lavoro. Eseguite l'applicazione facendo clic sul pulsante con la freccia gialla. Il nostro programma comparirà al centro dello schermo con il titolo "Main Frame". Ora chiudetelo facendo clic sulla "X" nell'angolo in alto a destra.

Richiamate il designer di FrameMain. Aggiungete due wx.Buttons, uno sopra l'altro e vicini al centro del frame. Selezionate quello superiore, chiamatelo "btnShowNew", e impostate l'etichetta a "Show the other frame" nella linguetta Constr della finestra Ispezionatore. Usate la combinazione Shift+Frecce per ridimensionare il pulsante così da rendere completamente visibile il testo, quindi usate la combinazione Ctrl+Frecce per muoverlo nuovamente al centro del frame. Selezionate il bottone inferiore, come Name usate "btnExit" e come label

"Exit". Fare clic su Invia, salvate ed eseguite per controllare i cambiamenti. Terminate il programma e ritornate al designer. È il momento di aggiungere gli eventi ai bottoni. Selezionate quello superiore e nel frame Ispezionatore selezionate la linguetta Evts. Fate clic su ButtonEvent, quindi doppio clic su wx.Evt_BUTTON. Notate che dovreste vedere "OnBtnShowNewButton" in basso. Successivamente, selezionate il pulsante btnExit. Fate la stessa cosa, assicurandovi che sia mostrato "OnBtnExitButton". Ancora clic su Invia e salvate. Quindi andate nella finestra Editor e scorrete verso il basso.

Assicuratevi che vi siano i due eventi appena creati. Ecco come dovrebbe apparire il frame:



Ora dobbiamo occuparci dell'altro frame. Aprite FrameSecond nel designer.

Impostate Name a "FrameSecond", e Title a "Second Frame". Impostare Centered a wx.BOTH. Aggiungete un wx.Button, allineandolo in basso al centro. Come Name inserire "btnFSExit" e cambiate il Title in "Exit". Impostate un evento. Quindi aggiungete un wx.StaticText nel mezzo della parte superiore del frame. Per Name mettete "stHiThere", per label "Hi there...I'm the second form!", e impostate il carattere a Sans, 14 punti e come weight mettete wxBOLD. Ora reimpostate la posizione in modo che sia centrato. Potete farlo deselezionando l'attributo Position e usare X per spostarlo a destra-sinistra e Y per spostarlo su-giù fino a quando non sarete soddisfatti. Fate clic su Invia e salvate:

Ora che abbiamo realizzato i nostri form, creeremo la "colla" che terrà tutto assieme.

Nella finestra Editor, fate clic sulla linguetta GUI2, quindi, al di sotto, fate clic sulla linguetta Sorgente. Sotto la riga che dice "import FrameMain", aggiungere "import FrameSecond". Salvate. Successivamente, selezionate la linguetta "FrameMain". Sotto la riga "import wx", aggiungete "import FrameSecond". Quindi scorrete in basso fino a trovare la riga con "def __init__(self, parent):". Aggiungete una riga dopo "self._init_ctrls(parent)" con scritto "self.Fs = FrameSecond.FrameSecond(self)". Ora sotto l'evento "def OnBtnShowNewButton(self, event):", commentate "event.Skip()" e aggiungete le seguenti due righe:

```
self.Fs.Show()
self.Hide()
```

Per finire, sotto il metodo "OnBtnExitButton", commentate "event.Skip()", e aggiungete la riga "self.Close()".

A cosa serve tutto ciò? Ok. La prima cosa che abbiamo fatto è assicurarci che l'applicazione sappia di avere due form. Ecco perché abbiamo importato sia FrameMain che FrameSecond in GUI2. Quindi abbiamo importato un riferimento per FrameSecond in FrameMain così da poterlo chiamare dopo. Lo abbiamo inizializzato nel metodo "_init_". E nell'evento "OnBtnShowNewButton" abbiamo specificato che quando si fa clic sul pulsante, vogliamo mostrare il secondo frame e nascondere quello principale. Quindi abbiamo l'istruzione per chiudere l'applicazione quando si fa clic su Exit.

Ora, passiamo al codice di FrameSecond. Qui le modifiche sono poche. Sotto il metodo "_init_", aggiungiamo la riga "self.parent =parent" che aggiunge la variabile self.parent. Infine, sotto l'evento di FSExitButton, commentate la riga "event.Skip()", e aggiungete queste due:

```
self.parent.Show()
self.Hide()
```

Ricordate che nascondiamo il frame principale quando mostriamo il secondo, per poi mostrarlo nuovamente dopo aver nascosto il secondo. Salvate.

Ecco il codice per verificare quanto fatto (questa e la pagina successiva):

Codice GUI2:

```
#!/usr/bin/env python
#Boa:App:BoaApp

import wx

import FrameMain
import FrameSecond

modules ={u'FrameMain': [1, 'Main frame of Application',
u'FrameMain.py'],
 u'FrameSecond': [0, '', u'FrameSecond.py']}

class BoaApp(wx.App):
    def OnInit(self):
        self.main = FrameMain.create(None)
        self.main.Show()
        self.SetTopWindow(self.main)
        return True

def main():
    application = BoaApp(0)
    application.MainLoop()

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

Codice FrameMain:

```
#Boa:Frame:FrameMain

import wx
import FrameSecond

def create(parent):
    return FrameMain(parent)

[wxID_FRAMEMAIN, wxID_FRAMEMAINBTNEXIT,
wxID_FRAMEMAINBTNSHOWNEW,
 wxID_FRAMEMAINPANEL1,
] = [wx.NewId() for _init_ctrls in range(4)]

class FrameMain(wx.Frame):
    def _init_ctrls(self, prnt):
        # generated method, don't edit
        wx.Frame.__init__(self, id=wxID_FRAMEMAIN,
name=u'FrameMain',
              parent=prnt, pos=wx.Point(846, 177),
size=wx.Size(400, 340),
              style=wx.DEFAULT_FRAME_STYLE, title=u'Main
Frame')
        self.SetClientSize(wx.Size(400, 340))
        self.Center(wx.BOTH)

        self.panel1 = wx.Panel(id=wxID_FRAMEMAINPANEL1,
name='panel1',
              parent=self, pos=wx.Point(0, 0),
size=wx.Size(400, 340),
              style=wx.TAB_TRAVERSAL)

        self.btnShowNew =
wx.Button(id=wxID_FRAMEMAINBTNSHOWNEW,
              label=u'Show the other frame',
name=u'btnShowNew',
              parent=self.panel1, pos=wx.Point(120,
103), size=wx.Size(168, 29),
              style=0)
        self.btnShowNew.SetBackgroundColour(wx.Colour(25,
 175, 23))
        self.btnShowNew.Bind(wx.EVT_BUTTON,
self.OnBtnShowNewButton,
              id=wxID_FRAMEMAINBTNSHOWNEW)
```

Codice FrameMain (cont.):

```
        self.btnExit =
wx.Button(id=wxID_FRAMEMAINBTNEXIT, label=u'Exit',
              name=u'btnExit', parent=self.panel1,
pos=wx.Point(162, 191),
              size=wx.Size(85, 29), style=0)
        self.btnExit.SetBackgroundColour(wx.Colour(225,
218, 91))
        self.btnExit.Bind(wx.EVT_BUTTON,
self.OnBtnExitButton,
              id=wxID_FRAMEMAINBTNEXIT)

    def __init__(self, parent):
        self._init_ctrls(parent)
        self.Fs = FrameSecond.FrameSecond(self)

    def OnBtnShowNewButton(self, event):
        #event.Skip()
        self.Fs.Show()
        self.Hide()

    def OnBtnExitButton(self, event):
        #event.Skip()
        self.Close()
```

Codice FrameSecond:

```
#Boa:Frame:FrameSecond

import wx

def create(parent):
    return FrameSecond(parent)

[wxID_FRAMESECOND, wxID_FRAMESECONDBTNFSEXIT,
wxID_FRAMESECONDPANEL1,
 wxID_FRAMESECONDSTATICTEXT1,
] = [wx.NewId() for _init_ctrls in range(4)]

class FrameSecond(wx.Frame):
    def _init_ctrls(self, prnt):
        # generated method, don't edit
        wx.Frame.__init__(self, id=wxID_FRAMESECOND,
name=u'FrameSecond',
              parent=prnt, pos=wx.Point(849, 457),
size=wx.Size(419, 236),
              style=wx.DEFAULT_FRAME_STYLE, title=u'Second
Frame')
        self.SetClientSize(wx.Size(419, 236))
        self.Center(wx.BOTH)
        self.SetBackgroundStyle(wx.BG_STYLE_COLOUR)

        self.panel1 = wx.Panel(id=wxID_FRAMESECONDPANEL1,
name='panel1',
              parent=self, pos=wx.Point(0, 0),
size=wx.Size(419, 236),
              style=wx.TAB_TRAVERSAL)

        self.btnFSExit =
wx.Button(id=wxID_FRAMESECONDBTNFSEXIT, label=u'Exit',
              name=u'btnFSExit', parent=self.panel1,
pos=wx.Point(174, 180),
              size=wx.Size(85, 29), style=0)
        self.btnFSExit.Bind(wx.EVT_BUTTON,
self.OnBtnFSExitButton,
              id=wxID_FRAMESECONDBTNFSEXIT)

        self.staticText1 =
wx.StaticText(id=wxID_FRAMESECONDSTATICTEXT1,
              label=u"Hi there...I'm the second form!",
name='staticText1',
              parent=self.panel1, pos=wx.Point(45, 49),
size=wx.Size(336, 23),
              style=0)
        self.staticText1.SetFont(wx.Font(14, wx.SWISS,
wx.NORMAL, wx.BOLD,
              False, u'Sans'))

    def __init__(self, parent):
        self._init_ctrls(parent)
        self.parent = parent

    def OnBtnFSExitButton(self, event):
        #event.Skip()
        self.parent.Show()
        self.Hide()
```

Ora potete eseguire l'applicazione. Se tutto è esatto, facendo clic su "Show the other frame" vedremo il primo frame scomparire e apparire il secondo. Facendo clic sul bottone Exit del secondo frame questo scomparirà e ricomparirà quello principale. Fate clic sul pulsante Exit del frame principale per chiudere l'applicazione.

Vi avevo promesso che avremmo parlato di convenzioni per i nomi. Ricordate quando parlammo di commenti al codice? Bene, scegliere con cura i nomi per i controlli della GUI è già un primo passo. È molto importante scegliere nomi significativi quando create frame complessi con molti controlli, specialmente se ci sono molti box di testo e pulsanti, invece di lasciare staticText1 o button1 o altro. Potrebbe non esserlo se sarete gli unici a controllare il codice, ma un altro programmatore troverebbe giovamento da nomi ben scelti. Quindi, usate qualcosa di simile ai seguenti:

```
Tipo controllo - Prefisso nome
Static text - st_
Button - btn_
Text Box - txt_
Check Box - chk_
Radio Button - rb_
Frame - Frm_ or Frame_
```

Potrete usare altri sistemi man mano che diventerete esperti, e in certi casi il vostro capo potrebbe già avere delle convenzioni proprie.

La prossima volta metteremo da parte la programmazione di GUI e ci concentreremo sui database. Nel frattempo, procuratevi python-apsw e python-mysqldb. Avrete anche bisogno di sqlite e sqlitebrowser per SQLite. È una buona idea sperimentare anche con MySql. Sono tutti disponibili via Synaptic.

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.

Un vero amico

di Richard Redei

L'intervista

di Richard Redei

Programmare in Python - Parte 7

Scritto da Greg Walters

VEDI ANCHE:
FCM nn. 27-32 - Python parti 1 - 6

VALIDO PER:
ubuntu kubuntu xubuntu

CATEGORIE:
Sviluppo Grafica Internet M/media Sistema

DISPOSITIVI:
CD/DVD HDD USB Drive Laptop Wireless

Buongiorno ragazzi e ragazze. È il momento di una storia. Siete tutti seduti comodamente? Pronti? Bene!

C'era una volta un mondo governato dalla carta. Carta, carta ovunque. C'era bisogno di ripostigli speciali per tutta quella carta. Erano chiamati schedari ed erano delle cose grandi di metallo che per conservare tutta quella carta occupavano stanze su stanze su stanze negli uffici. In ciascun schedario c'era qualcosa chiamato cartella, con lo scopo di organizzare insieme le carte attinenti. Ma col tempo, essi si riempivano e cadevano a pezzi quando diventavano vecchi o dopo essere stati aperti troppe volte.

Usare correttamente uno di questi schedari richiedeva una laurea. Poteva richiedere giorni trovare tutte le carte nei vari schedari. Il lavoro ne soffriva terribilmente. È stato davvero un periodo nero nella soria dell'umanità.

Poi un giorno, dalla cima di una montagna in un luogo imprecisato (penso si tratti del Colorado, ma non ne sono sicuro), arrivò un'adorabile fata. Questa fata era blu e argento con bellissime ali e capelli bianchi, ed era alta circa 30 centimetri. Il suo nome, che ci crediate o no, era See-Quill. Non è un nome simpatico? Comunque, See-Quill affermava di poter risolvere il problema della carta, degli schedari e del tempo sprecato, se solo le persone avessero creduto nei computer e in lei. Chiamò questo potere "Database". Disse che il "Database" poteva sostituire tutto il sistema di archiviazione. Alcune persone lo fecero e subito le loro vite divennero felici. Alcuni non seguirono il consiglio e la loro vita rimase identica, perduta tra montagne di carta.

Tutte le promesse della fata, pero', richiedevano una certa condizione. Questa condizione consisteva nel fatto che chiunque avesse voluto usare il potere di See-Quill doveva imparare un po' di un nuovo linguaggio. Non sarebbe stato troppo difficile imparare il linguaggio. Infatti, era molto simile a quello che le persone usavano. Aveva soltanto un modo differente di chiamare le cose, e si doveva pensare alle cose molto attentamente PRIMA di chiamarle - per usare il potere di See-Quill.

Un giorno, un giovanotto chiamato, abbastanza curiosamente, Utente, andò da See-Quill. Rimase molto impressionato dalla sua bellezza, e disse "See-Quill, per favore insegnami ad usare il tuo potere.". See-Quill acconsentì.

Disse, "Per prima cosa, devi sapere come le tue informazioni sono organizzate. Mostrami le tue carte."

Essendo giovane, Utente aveva solo pochi fogli di carta. See-Quill disse, "Utente, per ora puoi vivere con carte e cartelle. Però posso vedere il futuro e un giorno avrai così tanta carta che se accatastata sarà più alta di te di 15 volte. Dovremo usare il mio potere.".

Così, lavorando insieme, Utente e See-Quill diedero vita a un "database qualcosa" (un fiabesco nome tecnico) e Utente visse felice per sempre.

Fine.

Naturalmente, la storia non è completamente vera. Comunque, l'uso del database e di SQL può facilitarci la vita. Questa volta impareremo alcune semplici query SQL e come usarle in un programma. Qualcuno potrebbe pensare che questo non sia il modo "corretto" o "migliore", ma è comunque ragionevole. Quindi iniziamo.

I database sono come gli schedari della nostra storia. Le tabelle sono come le cartelle. I singoli record delle tabelle sono come i fogli di carta. Ciascun pezzo di informazione è chiamato campo. Si incastra bene insieme, non è vero? Usate istruzioni SQL (pronunciato See-Quill) per fare cose con i dati. SQL sta per Structured Query Language, ed è essenzialmente concepito per usare facilmente i database. In pratica, però, puo' diventare molto complesso. Ci manterremo sul semplice in questa lezione.

Abbiamo bisogno di creare uno schema, come quando si inizia un progetto edilizio. Così pensate ad una ricetta, che risulta un buon esempio visto che stiamo per creare un database di ricette. A casa mia le ricette sono presenti in varie forme: schede 3x5, pezzi di carta 8x10, tovaglioli con su stampate ricette, pagine di riviste e forme ancora più strane. Possiamo trovarle sui libri, scatole, copertine e altre cose. Comunque tutte hanno in comune una cosa: il formato. Nella maggior parte dei casi all'inizio abbiamo il titolo della ricetta e probabilmente il numero di porzioni e la provenienza. La parte centrale contiene la lista degli ingredienti e in basso le istruzioni - l'ordine in cui si procede, il tempo di cottura, e così via. Useremo questo formato generale come modello per il progetto del nostro database. Lo divideremo in due parti. Questa volta creeremo il database e la prossima l'applicazione per leggere e aggiornare il database.

Ecco un esempio. Diciamo di avere la ricetta sulla destra.

Osservate l'ordine appena discusso. Progettando il nostro database lo possiamo fare molto grande e avere un record per ciascun elemento della ricetta. In questa maniera, però, risulterebbe rozzo e difficile da gestire. Invece, useremo la scheda della ricetta come modello. Una tabella si occuperà dell'inizio della scheda, o delle informazioni principali della ricetta; una conterrà la parte centrale, o le informazioni sugli ingredienti; ed una per la parte finale, o le istruzioni.

Assicuratevi di aver installato SQLite e APSW. SQLite è un piccolo motore del database che non richiede un database server separato, il che lo rende ideale per la nostra piccola applicazione. Tutto quello che imparerete qui potrà essere usato con sistemi database più grandi come MySQL e altri. L'altro aspetto positivo di SQLite è che usa pochi tipi di dati. Questi sono Testo, Numerico, Blob e Chiave Primaria Intera. Come già sapete, testo puo' essere

Riso Spagnolo

Porzioni: 4

Fonte: Greg Walters

Ingredienti:
1 tazza di riso parboiled (cioè non cotto)
500 grammi di manzo tritato
2 tazze di acqua
225 g di salsa di pomodoro
1 piccola cipolla tritata
1 spicchio d'aglio tritato
1 cucchiaio da tavola di cumino
1 cucchiaio da tavola di origano
Sale e pepe quanto basta
Salsa a piacere

Istruzioni:
Rosolare la carne.

Aggiungere gli altri ingredienti.

Portare a ebollizione.

Mescolare, cuocere a fuoco lento e coprire.

Cucinare per 20 minuti.

Non guardare, non toccare.

Mescolare e servire.

qualunque cosa. I nostri ingredienti, le istruzioni e il titolo della nostra ricetta sono tutti di tipo testo - anche se contengono numeri. I tipi dato numerico sono numeri. Possono essere valori interi o a virgola mobile o reali. I Blob sono dati binari e possono comprendere immagini e altre cose. I valori di tipo Chiave Primaria Intera sono speciali. Il motore di SQLite assegna automaticamente per noi un valore intero con la garanzia di essere univoco. Sarà importante in seguito.

APWS sta per Another Python SQLite Wrapper ed è un sistema veloce per comunicare con SQLite. Ora esaminiamo alcuni possibili modi di creare le nostre istruzioni SQL.

Per recuperare un record dal database, userete l'istruzione SELECT. Il formato sarà:

```
SELECT [cosa] FROM [quale(i) tabella(e)] WHERE [restrizioni]
```

Così se vogliamo prendere tutti i campi dalla tabella Ricette useremo:

```
SELECT * FROM Ricette
```

Se desiderate ottenere solo un record dalla sua chiave primaria, dovete conoscere il suo valore (pkID in questo esempio), e dobbiamo includere il comando WHERE nell'istruzione. Potremmo usare:

```
SELECT * FROM Ricette WHERE pkID = 2
```

Abbastanza semplice... vero? Un linguaggio molto chiaro. Ora, supponiamo di voler recuperare solo il nome della ricetta e il numero di porzioni - per tutte le ricette. È facile. Tutto quello che dovete fare è includere una lista di campi che volete nell'istruzione SELECT:

```
SELECT nome,porzioni FROM Ricette
```

Per inserire dei record useremo il comando INSERT INTO. La sintassi è

```
INSERT INTO [nome tabella] (lista campi) VALUES (valori da inserire)
```

Così, per inserire una ricetta nella tabella delle ricette il comando sarà

```
INSERT INTO Ricette (nome,porzioni,provenienza) VALUES ("Tacos",4,"Greg")
```

Per cancellare un record possiamo usare

```
DELETE FROM Ricette WHERE pkID = 10
```

Esiste anche l'istruzione UPDATE, ma la lasceremo per un'altra volta.

Ancora su SELECT

Nel nostro esempio abbiamo tre tabelle, che possono essere relazionate usando ricettaID che punta a pkID della tabella ricette. Diciamo di voler recuperare tutte le istruzioni per una data ricetta. Lo possiamo fare così:

```
SELECT Ricette.nome,Ricette.porzioni,Ricette.provenienza,Istruzioni.Istruzioni FROM Ricette LEFT JOIN Istruzioni ON (Ricette.pkID = Istruzioni.ricettaID) WHERE Ricette.pkID = 1
```

Però si tratta di scrivere molto e con ridondanza. Possiamo usare un metodo chiamato aliasing. Lo possiamo fare così:

```
SELECT r.nome, r.porzioni,r.provenienza,i.Istruzioni FROM Ricette r LEFT JOIN Istruzioni i ON (r.pkID = i.ricettaID) WHERE r.pkID = 1
```

È più corta e ancora leggibile. Ora scriveremo un breve programma che creerà il nostro database, le nostre tabelle e inserirà qualche semplice dato nelle tabelle con cui lavorare. POTREMMO scriverlo internamente al nostro programma ma, per questo esempio, lo faremo in un programma separato. Si tratta di un programma ad unica esecuzione - se proverete ad eseguirlo una seconda volta fallirà durante la creazione della tabella. Ancora, potremmo includerlo in un'istruzione try...catch, ma lo faremo un'altra volta.

Iniziamo importando il wrapper APSW.

```
import apsw
```

Quindi abbiamo bisogno di creare una connessione con il nostro database. Sarà salvata nella stessa cartella del programma. Quando creiamo questa connessione, SQLite verifica automaticamente l'esistenza del database. Quindi lo apre se già presente, altrimenti lo crea per noi. Una volta stabilita la connessione, abbiamo bisogno del cosiddetto cursore. Si crea un meccanismo utile per lavorare con il database. Quindi ricordate, abbiamo bisogno sia della connessione che del cursore. Entrambi sono creati così:

```
# Aprire/creare il database

connessione=apsw.Connection("
cookbook1.db3")
cursore=connection.cursor()
```

OK, abbiamo la nostra connessione e il nostro cursore. Ora dobbiamo creare le nostre tabelle. Ce ne saranno tre nel nostro programma. Una che contiene le informazioni sulla quantità, una le istruzioni e un'altra la lista degli ingredienti di ciascuna ricetta. Non potremmo farlo con una sola tabella? Beh, si, ma, come vedrete, risulterebbe una tabella molto grande e con un mucchio di informazioni duplicate.

```
RICETTE
------------
pkID (Integer Primary Key)
nome (Text)
fonte (Text)
porzioni (Text)
```

```
ISTRUZIONI
----------------------
pkID(Integer Primary Key)
ricettaID (Integer)
istruzioni (Text)
```

```
INGREDIENTI
--------------------
pkID (Integer Primary Key)
recipeID (Integer)
ingredienti (Text)
```

Possiamo considerare una struttura come quella sopra: ciascuna colonna è una tabella separata.

Ogni tabella ha un campo chiamato pkID. È la chiave primaria unica all'interno della tabella. È importante perché evita che due record siano completamente identici. È di tipo intero ed è assegnata automaticamente dal motore del database. Ne potete fare a meno? Certo, ma correte il rischio di creare record duplicati. Nel caso della tabella Ricette, useremo questo numero come riferimento per quale istruzione e lista ingredienti associare alla ricetta.

Metteremo prima di tutto l'informazione nel database cosicché nome, provenienza e porzioni vadano nella tabella ricette. Il pkID è assegnato automaticamente. Assicuriamoci che sia davvero il primo record della nostra tabella, affinché il motore del database assegni il valore 1 al pkID. Useremo questo valore per collegare l'informazione nelle altre tabelle a questa ricetta. La tabella istruzioni è facile. Contiene semplicemente il testo delle istruzioni, il proprio pkID e quindi un puntatore alla ricetta nella tabella ricette. La tabella ingredienti è un po' più complicata poiché abbiamo un record per ciascun ingrediente con il proprio pkID e un puntatore verso la tabella ricette.

Quindi, per creare la tabella ricette, definiamo una variabile di tipo stringa chiamata sql e le assegnamo il comando per creare la tabella:

```
sql = 'CREATE TABLE Ricette
(pkiD INTEGER PRIMARY KEY,
nome TEXT, porzioni TEXT,
provenienza TEXT)'
```

Poi dobbiamo dire a APSW di eseguire effettivamente il comando sql:

```
cursore.execute(sql)
```

Ora creiamo le altre tabelle:

```
sql = 'CREATE TABLE
Istruzioni (pkID INTEGER
PRIMARY KEY, istruzioni
TEXT, ricettaID NUMERIC)'

cursore.execute(sql)

sql = 'CREATE TABLE
Ingredienti (pkID INTEGER
PRIMARY KEY, ingredienti
TEXT, ricettaID NUMERIC)'
```

```
cursore.execute(sql)
```

Finito di creare le tabelle, useremo il comando INSERT INTO per inserire i dati in ciascuna tabella.

Ricordate, pkID è inserito automaticamente così che non lo includeremo tra i campi nelle istruzioni di inserimento. Poiché useremo i loro nomi possiamo elencare i campi in qualunque ordine, non necessariamente in quello usato durante la creazione. Finché conosciamo il loro nome, tutto funzionerà bene. L'istruzione per l'inserimento nella tabella ricette diventa

```
INSERT INTO Ricette (nome,
porzioni, provenienza)
VALUES ("Riso
Spagnolo",4,"Greg Walters")
```

Successivamente dobbiamo trovare il valore assegnato a pkID. Possiamo farlo con un semplice comando:

```
SELECT last_insert_rowid()
```

Però, il risultato non è di molta utilità. Dobbiamo usare una serie di istruzioni come le seguenti:

```
sql = "SELECT
last_insert_rowid()"

cursore.execute(sql)

for x in
cursore.execute(sql):
    ultimoid = x[0]
```

Perché questo? Bene, quando APSW restituisce i dati lo fa sotto forma di tupla. Non ne abbiamo ancora parlato. La spiegazione rapida è che la tupla è (se guardate il codice sopra) come una lista, ma non modificabile. Alcune persone usano raramente le tuple, altre spesso; sta a voi decidere. L'ultima riga indica che vogliamo usare il primo valore restituito. Usiamo il ciclo 'for' per ottenere il valore contenuto nella variabile tupla x. Ha senso? OK, continuiamo...

Quindi, creeremo l'istruzione d'inserimento per le istruzioni:

```
sql = 'INSERT INTO
Istruzioni (ricettaID,
istruzioni) VALUES(
%s,"Rosolare la carne.
Aggiungere tutti gli altri
ingredienti. Portare a
ebollizione. Mescolare.
Cuocere a fuoco lento.
Coprire e cucinare per 20
minuti o fino alla completa
evaporazione del brodo.")' %
ultimoid

cursore.execute(sql)
```

Notate che stiamo usando la sostituzione di variabile (%s) per inserire il pkID della ricetta (ultimoid) nell'istruzione sql. Per finire, dobbiamo inserire ciascun ingrediente nella tabella ingredienti. Per il momento ve ne mostrerò solo uno:

```
sql = 'INSERT INTO
Ingredienti
(ricettaID,ingredienti)
VALUES ( %s,"1 tazza di riso
parzialmente cotto (non
cotto)")' % ultimoid

cursore.execute(sql)
```

Fino ad ora è stato semplice da capire. La prossima volta le cose si complicheranno un po'.

Se volete l'intero codice sorgente, lo trovate sul mio sito web. Visitate www.thedesignatedgeek.com per scaricarlo.

La prossima volta useremo quello che abbiamo imparato nelle ultime lezioni per creare un'interfaccia grafica per il nostro programma di ricette - ci permetterà di vedere tutte le ricette sotto forma di lista, di vedere una singola ricetta, cercarne una e aggiungerne o eliminarne.

Vi consiglio di dedicare un po' di tempo a leggere qualcosa sulla programmazione SQL. Non lo rimpiangerete.

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

Programmare In Python - Parte 8

Scritto da Greg Walters

VEDI ANCHE:
FCM#27-33 - Python Parti 1 - 7

VALIDO PER:
ubuntu kubuntu xubuntu

CATEGORIE:
Sviluppo Grafica Internet M/media Sistema

DISPOSITIVI:
CD/DVD HDD USB Drive Laptop Wireless

Riprendiamo il nostro database di ricette che abbiamo iniziato nella Parte 7. Questa sarà molto lunga, con tanto codice, quindi raccogliete le forze e non arrendetevi. Allacciate le cinture di sicurezza: si parte! Abbiamo già creato il nostro database. Ora vogliamo mostrarne il contenuto, fare aggiunte o rimozioni. Allora, come procediamo? Inizieremo con un'applicazione che gira nel terminale, quindi abbiamo bisogno di creare un menu.

Creeremo anche una classe che conterrà le nostre routine. Iniziamo con la parte del nostro programma mostrata in alto a destra.

Ora definiremo il nostro menu. Lo facciamo per poter abbozzare la nostra classe. Il nostro menu sarà un enorme ciclo che mostrerà una lista di opzioni che l'utente può scegliere. Useremo un ciclo while. Cambiate la funzione menu affinché appaia come mostrato in basso a destra.

Quindi aggiungeremo al menu una struttura if|elif|else che è mostrata in alto nella pagina seguente.

Diamo un'occhiata alla nostra routine menu. Iniziamo stampando le azioni che l'utente può eseguire. Impostiamo una variabile (loop) a True e quindi usiamo la funzione while per continuare il ciclo fino a quando loop = False. Useremo il comando raw_input() per aspettare che

```
#!/usr/bin/python
#-------------------------------------------------
# Ricettario.py
# Creato per Programmare in Python #8
# e Full Circle Magazine
#-------------------------------------------------
import apsw
import string
import webbrowser

class Ricettario:

def Menu():
    cbk = Ricettario() # Inizializza la classe

Menu()
```

```
def Menu():
    cbk = Ricettario() # Inizializza la classe
    loop = True
    while loop == True:
        print
'================================================='
        print '                DATABASE RICETTE'
        print
'================================================='
        print ' 1 - Mostra tutte le ricette'
        print ' 2 - Cerca una ricetta'
        print ' 3 - Mostra una ricetta'
        print ' 4 - Elimina una ricetta'
        print ' 5 - Aggiungi una ricetta'
        print ' 6 - Stampa una ricetta'
        print ' 0 - Esci'
        print
'================================================='
        response = raw_input('Inserisci una selezione -> ')
```

```
        if risposta == '1': # Mostra tutte le ricette
            pass
        elif risposta == '2': # Cerca una ricetta
            pass
        elif risposta == '3': # Mostra una ricetta
            pass
        elif risposta == '4': # Elimina una ricetta
            pass
        elif risposta == '5': # Aggiungi una ricetta
            pass
        elif risposta == '6': # Stampa una ricetta
            pass
        elif risposta == '0': # Termina il programma
            print 'Arrivederci'
            loop = False
        else:
            print 'Comando non riconosciuto. Riprova.'
```

l'utente scelga un'opzione e quindi la nostra routine if gestirà qualunque scelta fatta. Prima di testarla, dobbiamo inserire nella nostra classe la funzione init:

```
def __init__(self):
    pass
```

Ora, salvate il programma dove avete salvato il database creato l'ultima volta ed eseguitelo. Dovreste vedere qualcosa di simile a quello mostrato in alto a destra.

Dovrebbe semplicemente stampare il menu a oltranza finché non digitate "0", quindi stamperà "Arrivederci" ed uscirà. A questo punto ci possiamo occupare delle funzioni nella nostra classe Ricettario. Avremo bisogno di una funzione che mostri tutte le informazioni della tabella Ricette, di una che permetta di cercare una ricetta, di una che mostri i dati di una singola ricetta estratti da tutte e tre le tabelle, di una che cancelli una ricetta, di una che permetta di aggiungerla e di una che la stampi con la stampante predefinita. La funzione MostraTutteRicette non necessita di altri parametri oltre a quello (self), come neanche CercaRicetta e Inserisci Nuova. Le

```
/usr/bin/python -u
 "/home/greg/python_examples/APSW/cookbook/cookbook_stub.py"
==========================================================
                DATABASE RICETTE
==========================================================
1 - Mostra tutte le ricette
2 - Cerca una ricetta
3 - Mostra una ricetta
4 - Elimina una ricetta
5 - Aggiungi una ricetta
6 - Stampa una ricetta
0 - Esci
==========================================================
Inserire una selezione ->
```

funzioni MostraRicetta, EliminaRicetta e Stampa hanno tutte bisogno di conoscere la ricetta su cui operare, così necessiteranno del parametro che chiameremo "quale". Usate il comando pass per terminare ciascun blocco. Sotto la classe Ricettario create le funzioni:

```
def MostraTutteRicette(self):
  pass
def CercaRicetta(self):
  pass
def MostraRicetta(self,quale):
  pass
def
EliminaRicetta(self,quale):
  pass
def InserisciNuova(self):
  pass
def Stampa(self,quale):
  pass
```

Alcune delle voci di menu mostreranno tutte le ricette dalla tabella Ricette, cosicché l'utente ne possa scegliere una. Si tratta delle opzioni 1, 3, 4 e 6. Quindi modificate la routine menu in corrispondenza di queste opzioni, sostituendo il comando pass con cbk.MostraTutteRicette(). La funzione che analizza la scelta sarà come il codice in alto nella pagina successiva.

Un'altra cosa da fare è impostare la funzione init. Usate le righe seguenti:

```
def __init__(self):
  global connessione
  global cursore
  self.totalcount = 0
  connessione=apsw.Connection(
"cookbook.db3")
  cursore=connessione.cursor()
```

Prima creiamo due variabili globali per connessione e

```
if risposta == '1': # Mostra tutte le ricette
    cbk.MostraTutteRicette()
elif risposta == '2': # Cerca una ricetta
    pass
elif response == '3': # Mostra una ricetta
    cbk.MostraTutteRicette()
elif response == '4': # Elimina una ricetta
    cbk.MostraTutteRicette()
elif response == '5': # Aggiungi una ricetta
    pass
elif response == '6': # Stampa una ricetta
    cbk.MostraTutteRicette()
elif response == '0': # Termina il programma
    print 'Arrivederci'
    loop = False
else:
    print 'Comando non riconosciuto. Riprova.'
```

cursore. Potremo accedervi da ovunque all'interno della nostra classe ricettario. Poi, creiamo la variabile self.totricette che useremo per contare il numero di ricette. La useremo più avanti. Quindi creiamo la connessione e il cursore.

Il passo successivo sarà di rimpolpare la funzione MostraTutteRicette(). Poiché connessione e cursore sono variabili globali, non abbiamo bisogno di ricrearle in ciascuna routine. Ora vogliamo che le intestazioni della lista ricette vengano stampate a schermo in maniera elegante.

Useremo il comando di formattazione "%s" e il comando per giustificare a sinistra per spaziare il testo stampato a schermo. Vogliamo che appaia come questo:

```
Item Nome Porzioni Provenienza

----------------------------
```

Per finire dobbiamo creare l'istruzione SQL, interrogare il database e mostrare il risultato. La maggior parte sono state trattate nel precedente articolo.

```
    sql = 'SELECT * FROM
Ricette'
    cntr = 0
    for x in
cursore.execute(sql):
        cntr += 1
        print '%s %s %s %s'
%(str(x[0]).rjust(5),x[1].ljus
t(30),x[2].ljust(20),x[3].ljus
t(30))
    print '-------------'
    self.totricette = cntr
```

La variabile cntr conterrà il numero di ricette mostrate all'utente. La nostra funzione è pronta. In basso è mostrato tutto il codice, in caso vi sia sfuggito qualcosa.

Notate come si sia usata la tupla restituita dalla funzione cursore.execute di ASPW. Stampiamo pkID come l'item di ciascuna ricetta. Questo ci permetterà in seguito di selezionare la ricetta corretta.

Quando eseguite il programma vedrete il menu e scegliendo l'opzione 1 vedrete quello mostrato in alto nella pagina seguente.

Proprio come si voleva il programma non andrà in pausa, eccetto se lo eseguite in Dr.Python o simili. Aggiungiamo una pausa fino a quando l'utente non preme un tasto così che possa osservare l'output per uno o due secondi. Mentre siamo in questa fase stampiamo il numero di ricette dalla variabile impostata poco fa. Aggiungete alla fine dell'opzione 1 del menu:

```
print 'Ricette totali - %s'
%cbk.totricette
```

```
def MostraTutteRicette(self):
        print '%s %s %s %s'
%('Item'.ljust(5),'Nome'.ljust(30),'Porzioni'.ljust(20
),'Provenienza'.ljust(30))
        print '--------------------------------'
        sql = 'SELECT * FROM Ricette'
        cntr = 0
        for x in cursore.execute(sql):
            cntr += 1
            print '%s %s %s %s'
%(str(x[0]).rjust(5),x[1].ljust(30),x[2].ljust(20),x[3
].ljust(30))
        print '--------------------------------'
        self.totricette = cntr
```

```
Inserire una selezione -> 1
Item  Nome                              Porzioni                Provenienza
-------------------------------------------------------------------------------
    1 Spanish Rice                      4                       Greg
    2 Pickled Pepper-Onion Relish       9 half pints            Complete Guide to Home Canning
-------------------------------------------------------------------------------
==================================================
              DATABASE RICETTE
==================================================
1 - Mostra tutte le ricette
2 - Cerca una ricetta
3 - Mostra una ricetta
4 - Elimina una ricetta
5 - Aggiungi una ricetta
6 - Stampa una ricetta
0 - Esci
==================================================
Inserire una selezione ->
```

```
 print '----------------------
------------------------------
'

res = raw_input('Premere
Tasto A -> ')
```

Per il momento salteremo l'opzione 2 (Cerca una ricetta) e ci occuperemo della 3 (Mostra ricetta). Prima la parte riguardante il menu. Mostreremo la lista delle ricette, come nell'opzione 1 e quindi chiederemo di sceglierne una. Per evitare errori da un errato inserimento dell'utente useremo la struttura Try|Except. Mostreremo la richiesta all'utente (Seleziona la ricetta →), quindi se inserisce una scelta valida chiameremo MostraRicetta() con il pkID dalla tabella Ricette. Se non si è inserito un numero verrà avviata l'eccezione ValueError, usata come mostrato a destra.

Ora lavoriamo sulla funzione MostraRicetta. Iniziamo nuovamente con la connessione e il cursore quindi creiamo l'istruzione SQL. In questo caso usiamo "'SELECT * FROM Ricette WHERE pkID = %s' % str(quale)" dove quale è il valore che vogliamo trovare. Quindi abbelliamo l'output con la tupla ritornata da ASPW. In questo caso usiamo x come variabile cumulativa e usiamo indici tra parentesi per accedere ai singoli elementi della tupla. Dato che l'organizzazione della tabella è pkID/nome/porzioni/provenienza, possiamo usare x[0],x[1],x[2] e x[3] come dettaglio. Quindi, selezioniamo tutto dalla tabella ingredienti dove ricettaID (la nostra chiave nella tabella ricette) è uguale al pkID appena usato. Cicliamo nella tupla ritornata, stampando ciascun ingrediente e alla fine le istruzioni dalla tabella istruzioni, proprio come fatto per la tabella ingredienti. Per finire attendiamo che l'utente prema un tasto così che possa vedere la ricetta a schermo. Il codice lo trovate nella prossima pagina.

Bene, due delle sei funzioni sono pronte. Allora occupiamoci della funzione cerca, iniziando nuovamente dal menu. Fortunatamente questa volta chiameremo semplicemente la routine di cerca della classe,

```
try:
    res = int(raw_input('Seleziona una ricetta -> '))
    if res <= cbk.totricette:
        cbk.MostraRicetta(res)
    elif res == cbk.totricette + 1:
        print 'Torno al menu...'
    else:
        print 'Comando non riconosciuto. Torno al menu.'
except ValueError:
    print 'Non un numero. Torno al menu.'
```

quindi sostituite il comando pass con:

```
cbk.CercaRicetta()
```

Ora concretizziamo il nostro codice di ricerca. Nella classe Ricettario, sostituite il codice per CercaRicetta con quello mostrato nella pagina seguente.

C'è molto da fare. Dopo aver creato la connessione e il cursore mostriamo il menu di ricerca. Daremo all'utente tre modalità di ricerca più un'opzione per uscire. Possiamo consentire una ricerca sul nome della ricetta, sulla provenienza o nella lista ingredienti. Per questo non possiamo usare semplicemente la funzione appena creata per mostrare le ricette ma abbiamo bisogno di una nuova routine. Le prime due opzioni usano semplici istruzioni SELECT con un'aggiunta. Useremo il qualificatore "like". Se dovessimo usare un programma come SQLite Database Browser, la nostra istruzione like userebbe il carattere jolly "%". Così, per cercare una ricetta contenente nel nome la parola "riso", la nostra query sarebbe:

```
SELECT * FROM Ricette WHERE
nome like '%riso%'
```

Però, dato che il carattere "%" è anche un carattere di sostituzione nelle nostre stringhe, dobbiamo usare %% nel codice. Per ingarbugliare le cose useremo il carattere di sostituzione per inserire la parola che l'utente cerca. Per questo dovremo scrivere "%%%s%%". Scusate se è poco chiaro. La terza query è chiamata istruzione Join. Guardiamola più da vicino:

```
sql = "SELECT
r.pkid,r.nome,r.porzioni,r.pro
venienza,i.ingredienti FROM
Ricette r Left Join
ingredienti i on (r.pkid =
i.ricettaid) WHERE
i.ingredienti like '%%%s%%'
GROUP BY r.pkid" %risposta
```

Si procede selezionando tutto dalla tabella ricette e gli ingredienti da quella ingredienti, unendo o correlando la tabella Ingredienti SU ricettaID essendo questo uguale a pkID nella tabella ricette, quindi si cercano i nostri ingredienti usando l'istruzione like, e, finalmente, si raggruppano i risultati per pkID nella tabella ricette per evitare di mostrare doppioni. Se ricordate, abbiamo peperoni due volte nella seconda ricetta (Cipolla e peperoni gustosi), uno verde e uno rosso. Questo potrebbe creare confusione nell'utente. Il nostro menu usa

```
  def MostraRicetta(self,which):
        sql = 'SELECT * FROM Ricette WHERE pkID = %s' %
str(which)
        print
'~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~'
        for x in cursore.execute(sql):
            ricettaid =x[0]
            print "Titolo: " + x[1]
            print "Porzioni: " + x[2]
            print "Provenienza: " + x[3]
        print
'~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~'
        sql = 'SELECT * FROM Ingredienti WHERE RicettaID
= %s' % icettaid
        print 'Lista Ingredienti:'
        for x in cursore.execute(sql):
            print x[1]
        print ''
        print 'Istruzioni:'
        sql = 'SELECT * FROM Istruzioni WHERE RicettaID
= %s' % ricettaid
        for x in cursore.execute(sql):
            print x[1]
        print
'~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~'
        resp = raw_input('Premere Tasto A -> ')
```

```
  cercain =
raw_input('Inserisci Tipo di
Ricerca -> ')
```

```
if cercain != '4':
```

che significa: se cercain (il valore inserito dall'utente) NON è uguale a 4 allora esegui le opzioni, se è 4 allora non fare nulla, passa oltre. Notate che ho usato "!=" come Non Uguale A invece di "<>". Entrambe le forme sono valide in Python 2.x. Però in Python 3.x darà un errore di sintassi. Tratteremo dei

```
    def CercaRicetta(self):
        # mostra il menu di ricerca
        print '------------------------------'
        print ' Cerca in'
        print '------------------------------'
        print ' 1 - Nome ricetta'
        print ' 2 - Provenienza ricetta'
        print ' 3 - Ingredienti'
        print ' 4 - Esci'
        cercain = raw_input('Inserire Tipo di Ricerca -> ')
        if cercain != '4':
            if cercain == '1':
                cerca = 'Nome Ricetta'
            elif cercain == '2':
                cerca = 'Provenienza Ricetta'
            elif cercain == '3':
                cerca = 'Ingredienti'
            parm = cercain
            risposta = raw_input('Cosa cerco in %s (nulla per uscire) -> ' % cerca)
            if parm == '1': # Nome Ricetta
                sql = "SELECT pkid,nome,provenienza,porzioni FROM Ricette WHERE nome like '%%%s%%'" %risposta
            elif parm == '2': # Provenienza Ricetta
                sql = "SELECT pkid,nome,provenienza,porzioni FROM Ricette WHERE provenienza like '%%%s%%'" %risposta
            elif parm == '3': # Ingredienti
                sql = "SELECT r.pkid,r.nome,r.porzioni,r.provenienza,i.ingredienti FROM Ricette r Left Join ingredienti
i on (r.pkid = i.ricettaid) WHERE i.ingredienti like '%%%s%%' GROUP BY r.pkid" %risposta
            try:
                if parm == '3':
                    print '%s %s %s %s %s'
%('Item'.ljust(5),'Nome'.ljust(30),'Porzioni'.ljust(20),'Provenienza'.ljust(30),'Ingredienti'.ljust(30))
                    print '-------------------------------------------------------------------------------'
                else:
                    print '%s %s %s %s' %('Item'.ljust(5),'Nome'.ljust(30),'Porzioni'.ljust(20),'Provenienza'.ljust(30))
                    print '-------------------------------------------------------------------------------'
                for x in cursore.execute(sql):
                    if parm == '3':
                        print '%s %s %s %s %s'
%(str(x[0]).rjust(5),x[1].ljust(30),x[2].ljust(20),x[3].ljust(30),x[4].ljust(30))
                    else:
                        print '%s %s %s %s' %(str(x[0]).rjust(5),x[1].ljust(30),x[3].ljust(20),x[2].ljust(30))
            except:
                print 'Errore'
            print '-------------------------------------------------------------------------------'
            inkey = raw_input('Premere un tasto')
```

cambiamenti in Python 3.x in un articolo futuro. Per ora, iniziate a usare "!=" per rendere il passaggio a Python 3.x più semplice. Finalmente possiamo mostrare, abbellito, ancora l'output. A destra osserviamo cosa vedrà l'utente.

Potete vedere come il programma stampi in un modo carino l'output. Ora, l'utente può tornare indietro al menu e usare l'opzione 3 per mostrare qualunque ricetta voglia vedere. Il prossimo passo è aggiungere ricette al nostro database. Nuovamente, dobbiamo aggiungere una riga alla nostra funzione menu per chiamare la routine InserisciNuova:

```
cbk.InserisciNuova()
```

Il codice da inserire in InserisciNuova() nella classe Ricettario è qui: http://pastebin.com/f1d868e63.

Iniziamo definendo una lista chiamata "ingg", che sta per ingredienti. Quindi chiederemo all'utente di inserire il titolo, la provenienza e le porzioni. Quindi entreremo in un ciclo che richiede ciascun ingrediente da

```
Inserire una selezione -> 2
------------------------------
Cerca in
------------------------------
1 - Nome Ricetta
2 - Provenienza Ricetta
3 - Ingredienti
4 - Esci
Inserire Tipo di Ricerca -> 1
Cosa cerco in Nome Ricetta (nulla per uscire) -> rice
Item  Nome                      Porzioni           Provenienza
--------------------------------------------------------------------------------
    1 Spanish Rice                    4                    Greg
--------------------------------------------------------------------------------
Premere un tasto
```

Abbastanza semplice. Ora la ricerca degli ingredienti...

```
Inserire una selezione -> 2
------------------------------
Cerca in
------------------------------
1 - Nome Ricetta
2 - Provenienza Ricetta
3 - Ingredienti
4 - Esci
Inserire Tipo Ricerca -> 3
Cosa cerco in Ingredienti (nulla per uscire) -> onion
Item  Nome                Porzioni      Provenienza                          Ingredienti

--------------------------------------------------------------------------------
1 Spanish Rice                  4              Greg                              1 small
Onion chopped
2 Pickled Pepper-Onion Relish  9 half pints   Complete Guide to Home Canning 6 cups
finely chopped Onions
--------------------------------------------------------------------------------
Premere un tasto
```

aggiungere alla lista ingg. Se l'utente inserisce 0, si esce dal ciclo e si continua chiedendo le istruzioni. Quindi mostriamo il contenuto della ricetta chiedendo all'utente di verificarlo prima di salvarlo. Useremo istruzioni INSERT INTO, come abbiamo fatto l'ultima volta, e ritorniamo al menu. Dobbiamo fare attenzione agli apici singoli nei nostri inserimenti. GENERALMENTE non saranno un problema nella lista ingredienti o nelle istruzioni, ma nei campi titolo e provenienza potrebbero esserlo. Bisogna aggiungere un carattere di escape prima di ogni apice singolo. Lo facciamo con la funzione string.replace che è il motivo per cui abbiamo importato la libreria string. Nell'opzione #4 della funzione menu inserire il codice in alto a destra.

Quindi, nella classe Ricettario usate il codice in basso a destra per la funzione EliminaRicetta().

Guardiamo velocemente la funzione per l'eliminazione. Prima chiediamo all'utente quale ricetta eliminare (torna al menu) e passiamo quel pkID nella nostra routine. Poi chiediamo all'utente 'se è SICURO' di voler eliminare la ricetta. Se la risposta è "S" (string.upper(resp) == 'S') allora creeremo le istruzioni sql per l'eliminazione. Notate come questa volta dobbiamo cancellare record in tutte e tre le tabelle. Potremmo solo cancellare quello della tabella ricette, ma ci ritroveremmo con record orfani nelle altre due, e non è un bene. Quando eliminiamo un record dalla tabella ricette, usiamo il campo pkID, nelle altre due tabelle usiamo ricettaID.

Per finire ci occupiamo della funzione per stampare le ricette. Creeremo un file HTML MOLTO semplice, lo apriremo nel browser predefinito per stamparlo. Ecco perché importiamo la libreria webbrowser. Nella funzione menu per l'opzione 6 inserite il codice mostrato in cima alla pagina successiva.

Ancora, mostreremo una lista di tutte le ricette per selezionare quella che si desidera stampare. Chiameremo la funzione Stampa nella classe Ricettario. Il codice è mostrato in alto a destra nella prossima pagina.

Iniziamo con il comando fi = open([nomefile],'w') che crea il file. Quindi prendiamo i dati dalla tabella ricette e li scriviamo nel file con il comando fi.write. Usiamo il tag intestazione 1 <H1></H1> per il titolo, quello <H2> per le porzioni e la provenienza. Usiamo quindi i tag

```
cbk.MostraTutteRicette()
     print '0 - Ritorna al Menu'
     try:
         res = int(raw_input('Seleziona una Ricetta da
ELIMINARE o 0 per uscire -> '))
         if res != 0:
             cbk.EliminaRicetta(res)
         elif res == '0':
             print 'Ritorno al Menu...'
         else:
             print 'Comando non riconosciuto.  Torno al
menu.'
     except ValueError:
          print 'Non un numero...torno al menu.'
```

```
    def EliminaRicetta(self,quale):
       resp = raw_input('Sei SICURO di voler ELIMINARE
questo record? (S/n) -> ')
        if string.upper(resp) == 'S':
            sql = "DELETE FROM Ricette WHERE pkID = %s" %
str(quale)
            cursore.execute(sql)
            sql = "DELETE FROM Istruzioni WHERE ricettaID
= %s" % str(quale)
            cursore.execute(sql)
            sql = "DELETE FROM Ingredienti WHERE ricettaID
= %s" % str(quale)
            cursore.execute(sql)
            print "Ricetta ELIMINATA"
            resp = raw_input('Premere il tasto A -> ')
        else:
            print "Eliminazione annullata - Torno al menu"
```

per l'elenco degli ingredienti e concludiamo con le istruzioni. Oltre questo si tratta di semplici query già imparate. Infine, chiudiamo il file con il comando fi.close() e usiamo webbrowser.open([nomefile]) con il file appena creato. L'utente può successivamente stamparlo dal proprio browser, se lo desidera.

FIUU! Fino ad oggi questo è stato il programma più impegnativo. Ho inserito tutto il codice sorgente (e il database dell'esempio se vi siete persi quello del mese scorso) sul mio sito web. Se non volete trascriverlo tutto o avete qualche problema allora visitate www.thedesignatedgeek.com per recuperare il codice.

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.

```
cbk.MostraTutteRicette()
        print '0 - Torna al Menu'
        try:
            res = int(raw_input('Seleziona una Ricetta da ELIMINARE o 0 per uscire ->
'))
            if res != 0:
                cbk.Stampa(res)
            elif res == '0':
                print 'Torno al Menu...'
            else:
                print 'Comando non riconosciuto. Torno al menu.'
        except ValueError:
            print 'Non un numero...torno al menu.'
```

```
    def Stampa(self,quale):
        fi = open('stamparicetta.html','w')
        sql = "SELECT * FROM Ricette WHERE pkID = %s" % quale
        for x in cursore.execute(sql):
            NomeRicetta = x[1]
            ProvenienzaRicetta = x[3]
            PorzioniRicetta = x[2]
        fi.write("<H1>%s</H1>" % NomeRicetta)
        fi.write("<H2>Provenienza: %s</H2>" % ProvenienzaRicetta)
        fi.write("<H2>Porzioni: %s</H2>" % PorzioniRicetta)
        fi.write("<H3> Lista Ingredienti: </H3>")
        sql = 'SELECT * FROM Ingredienti WHERE RicettaID = %s' % quale
        for x in cursore.execute(sql):
            fi.write("<li>%s</li>" % x[1])
        fi.write("<H3>Istruzioni:</H3>")
        sql = 'SELECT * FROM Istruzioni WHERE RicettaID = %s' % quale
        for x in cursore.execute(sql):
            fi.write(x[1])
        fi.close()
        webbrowser.open('stamparicetta.html')
        print "Fatto"
```