full circle

TẠP CHÍ ĐỘC LẬP CỦA CỘNG ĐỒNG NGƯỜI DÙNG UBUNTU LINUX

Số đặc biệt - chuyên đề lập trình

SỐ ĐẶC BIỆT
CHUYÊN ĐỀ LẬP TRÌNH

LẬP TRÌNH PYTHON
TẬP 1

full circle

TẠP CHÍ ĐỘC LẬP CỦA CỘNG ĐỒNG NGƯỜI DÙNG UBUNTU LINUX

Phần 1: Làm quen với Python

Phần 2: Làm quen với Python (tt)

Phần 3: Mô-đun và Hàm

Phần 4: Lớp trong Python

Phần 5: Lập trình giao diện đồ họa

Phần 6: Lập trình giao diện đồ họa (tt)

Phần 7: Lập trình cơ sở dữ liệu

Phần 8: Lập trình cơ sở dữ liệu (tt)

Nhóm Full Circle

Chủ biên - Ronnie Tucker
ronnie@fullcirclemagazine.org
Quản lí website - Rob Kerfia
admin@fullcirclemagazine.org
Quản lí truyền thông - Robert Clipsham
mrmonday@fullcirclemagazine.org

Biên tập và hiệu đính
Mike Kennedy
David Haas
Gord Campbell

Xin gửi lời cảm ơn đến công ty Canonical, nhóm quảng bá Ubuntu và nhiều nhóm biên dịch trên khắp thế giới.

Dự án biên dịch tạp chí Full Circle:
http://tapchifc.pbworks.com

LÀM THẾ NÀO Lập trình Python – Phần 1

Viết bởi Greg Walters

Giữa vô vàn ngôn ngữ lập trình hiện nay, Python là ngôn ngữ dễ học nhất. Python ra đời vào cuối thập niên 80 của thế kỷ trước và phát triển không ngừng. Nó mặc định được cài đặt sẵn trong hầu hết các bản phân phối Linux, và thường được người ta chú ý đến khi quyết định học một ngôn ngữ lập trình. Chúng ta sẽ làm quen với lập trình dòng lệnh trong bài viết này. Sắp tới chúng ta sẽ khám phá lập trình giao diện đồ họa GUI (Graphical User Interface). Bây giờ, hãy bắt tay vào xây dựng một ứng dụng đơn giản.

Chương trình đầu tiên

Hãy dùng một trình soạn thảo văn bản như gedit để gõ một vài dòng mã lệnh. Sau đó xem cách chúng hoạt động.

Gõ 4 dòng sau:

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

print 'Hello.  I am a python program.'

name = raw_input("What is your name? ")

print "Hello there, " + name + "!"
```

Lưu tập tin lại với tên "hello.py" vào nơi nào bạn muốn. Tôi đề nghị các bạn tạo một thư mục tên python_examples trong thư mục chính (home) và đặt nó vào trong. Ví dụ đơn giản này cho thấy lập trình Python dễ như thế nào. Trước khi có thể chạy chương trình, chúng ta phải thiết lập thuộc tính thực thi cho nó. Mở một cửa sổ dòng lệnh và chuyển dấu nhắc vào trong thư mục mà bạn vừa lưu tập tin rồi gõ:

```
chmod +x hello.py
```

Bây giờ hãy chạy chương trình:

```
greg@earth:~/python_examples
$ ./hello.py

Hello.  I am a python program.

What is your name? Ferd Burphel

Hello there, Ferd Burphel!

greg@earth:~/python_examples
$
```

Thật đơn giản phải không. Bây giờ, hãy nhìn lại từng dòng lệnh.

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

Dòng này báo cho hệ thống biết đây là chương trình viết bằng Python và dùng bộ thông dịch python mặc định để thực thi chương trình.

```
print 'Hello. I am a python program.'
```

Dòng này sẽ in ra câu "Hello. I am a python program." trong cửa sổ dòng lệnh.

```
name = raw_input("What is your name? ")
```

Cái này hơi phức tạp chút. Có hai phần trong dòng này. Đầu tiên là name =, và thứ hai là raw_input("What is your name? "). Chúng ta sẽ xem phần thứ hai trước. Lệnh raw_input sẽ in câu ("What is your name?") ra cửa sổ dòng lệnh và nó sẽ đợi người dùng (các bạn) nhập vào nội dung gì đó (kết thúc bằng phím {Enter}). Bây giờ hãy quay lại phần đầu: name =. Phần này sẽ gán giá trị vào một biến tên là "name". Thế biến là gì? Hãy tưởng tượng biến là một cái hộp dùng để chứa đồ, giày dép, linh kiện máy tính, giấy tờ, bất kể thứ gì. Trong trường hợp này, nó chứa những gì bạn nhập. Vì vậy tôi đã nhập Ferd Burphel. Trong ví dụ này, Python chỉ đơn giản lấy dữ liệu nhập và cất vào chiếc hộp "name" để dùng sau này.

```
print "Hello there, " + name + "!"
```

Một lần nữa, chúng ta sử dụng câu lệnh "print" để hiển thị thông tin trên màn hình, đó là "Hello there, ", cộng với dữ liệu có trong biến "name", và một dấu chấm than ở cuối câu. Ở đây chúng ta nối hay nói cách khác là đặt gần nhau ba mảnh thông tin: "Hello there", thông tin trong biến "name" và

dấu chấm than.

Bây giờ, hãy dành một chút thời gian để tìm hiểu sâu hơn trước khi chúng ta đi đến một ví dụ khác. Mở cửa sổ dòng lệnh và gõ:

```
python
```

Các bạn sẽ thấy điều gì đó tương tự thế này:

```
greg@earth:~/python_examples$
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.

>>>
```

Các bạn đang ở trong giao diện shell của Python. Từ đây, các bạn có thể làm được nhiều điều, nhưng hãy xem chúng ta có gì trước khi đi tiếp. Đầu tiên, các bạn nên lưu ý số phiên bản python - ở đây là 2.5.2. Kế đến, các bạn thấy câu thông báo rằng để được giúp đỡ, hãy gõ "help" tại dấu nhắc. Tôi sẽ để các bạn tự làm điều đó. Bây giờ hãy nhập:

```
print 2+2
```

và nhấn enter. Các bạn sẽ nhận được

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

Lưu ý rằng chúng ta viết từ "print" dưới dạng chữ thường. Điều gì sẽ xảy ra nếu chúng ta viết thành "Print 2+2"? Câu trả lời của trình thông dịch sẽ là:

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

Bởi vì chỉ tồn tại lệnh "print" chứ không phải "Print". Phân biệt chữ hoa – thường rất quan trọng trong Python.

Bây giờ hãy vui đùa với biến thêm một chút nữa. Gõ:

```
var = 2+2
```

Bạn sẽ thấy không có gì xảy ra ngoại trừ Python trả về dấu nhắc ">>>". Không có gì sai đâu. Chúng ta muốn Python tạo một biến (chiếc hộp) tên là var và bỏ vào đấy tổng của "2+2". Để xem var chứa gì, gõ:

```
print var
```

và nhấn enter.

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

Bây giờ chúng ta có thể sử dụng var bao nhiêu lần tuỳ thích thay cho số 4, như thế này:

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

Nếu chúng ta gõ lại "print var" thì sẽ nhận được:

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

var chẳng thay đổi. Nó vẫn là tổng của 2+2, là 4.

Dĩ nhiên đây là chương trình đơn giản cho bài hướng dẫn mở đầu này. Độ phức tạp sẽ tăng dần trong những bài kế tiếp. Nhưng bây giờ hãy xem tiếp một vài ví dụ về biến.

Vẫn trong trình thông dịch, gõ:

```
>>> strng = 'The time has
come for all good men to come
to the aid of the party!'

>>> print strng

The time has come for all
good men to come to the aid
of the party!

>>>
```

Các bạn vừa tạo một biến tên "strng" (viết tắt của string) chứa giá trị 'The time has come for all good men to come to the aid of the party!'. Từ bây giờ (miễn là chúng ta đừng thoát khỏi bộ thông dịch này), biến strng sẽ giữ nguyên giá trị trừ phi chúng ta thay đổi nó. Điều gì sẽ xảy ra nếu chúng ta thử nhân biến này với 4?

```
>>> print strng * 4

The time has come for all
good men to come to the aid
of the party!The time has
come for all good men to come
to the aid of the party!The
time has come for all good
men to come to the aid of the
```

```
party!The time has come for
all good men to come to the
aid of the party!

>>>
```

Chao ôi, nó không đúng như những gì chúng ta mong đợi, phải không? Nó in giá trị của biến strng 4 lần. Tại sao? Trình thông dịch hiểu rằng strng là một chuỗi ký tự, không phải là một giá trị số. Các bạn không thể biểu diễn một phép toán với một chuỗi.

Điều gì sẽ xảy ra nếu chúng ta có một biến tên s chứa chữ số '4' (ký tự) như sau:

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

Có vẻ như s chứa một số nguyên 4, nhưng không phải vậy. Thực tế là nó chứa một chuỗi thể hiện chữ số 4. Vậy, nếu chúng ta gõ 'print s * 4' thì...

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

Một lần nữa, trình thông dịch hiểu rằng s là một chuỗi, không phải là giá trị số. Bởi vì chúng ta bọc số 4 trong cặp nháy đơn, biểu diễn một chuỗi.

Chúng ta có thể chứng minh điều này bằng cách gõ print type(s) để xem kiểu dữ liệu của biến.

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

Chính xác! Đó là một kiểu chuỗi. Nếu chúng ta muốn dùng nó như một giá trị số, ta sẽ làm như sau:

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

Chuỗi (s), chứa giá trị '4', giờ đã được chuyển thành số nguyên và được nhân với 4 cho kết quả 16.

Vừa rồi, các bạn đã được giới thiệu lệnh print, raw_input, cách gán biến và sự khác biệt giữa chuỗi và số nguyên.

Hãy cùng tôi đi xa thêm chút nữa. Trong trình thông dịch Python, gõ quit() để thoát ra ngoài dấu nhắc của cửa sổ dòng lệnh.

Vòng lặp For đơn giản

Bây giờ hãy khám phá cách lập trình một vòng lặp đơn giản. Quay lại trình soạn thảo văn bản và gõ đoạn mã sau:

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

for cntr in range(0,10):

    print cntr
```

Đừng quên dùng phím Tab để thụt lề trước dòng "print cntr". Điều này rất quan trọng, Python không dùng dấu ngoặc đơn "(" hoặc dấu ngoặc nhọn "{" như những ngôn ngữ lập trình khác để biểu thị khối mã lệnh. Thay vào đó, nó căn cứ vào sự thụt lề đầu dòng.

Lưu chương trình với tên "for_loop.py". Trước khi thử chạy nó, hãy tìm hiểu vòng lặp for là gì.

Một vòng lặp là một khối mã lệnh thực hiện một công việc cụ thể và kèm theo số lần thực hiện. Trong trường hợp này, chúng ta sẽ cho lặp 10 lần, in giá trị của biến cntr (viết tắt của counter). Diễn giải ra là "gán biến cntr bằng 0, thực hiện 10 lần việc in giá trị của cntr lên màn hình, cộng 1 vào cntr sau mỗi chu kỳ thực hiện." Đoạn mã "range(0,10)" nói rằng hãy bắt đầu ở 0, lặp cho đến khi giá trị của cntr là 10, rồi kết thúc vòng lặp.

Quay trở lại cửa sổ dòng lệnh, trước khi chạy, nhớ:

```
chmod +x for_loop.py
```

và bây giờ chạy chương trình

```
./for_loop.py

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

Xem, có vẻ nó đã chạy đúng, nhưng tại sao nó chỉ đếm đến 9 rồi dừng lại. Nhìn lại kết quả xuất ra. Có 10 số đã được in, bắt đầu từ 0 và kết thúc bằng 9. Đó là điều chúng ta yêu

cầu máy làm - in giá trị của cntr 10 lần, thêm 1 vào biến mỗi lần, và kết thúc khi giá trị bằng 10.

Các bạn có thể thấy rằng, việc lập trình trông đơn giản, những cũng phức tạp không kém và các bạn cần biết chắc mình yêu cầu máy làm gì. Nếu bạn thay đổi câu lệnh range thành "range(1,10)", nó sẽ bắt đầu đếm từ 1, nhưng kết thúc bằng 9, vì khi cntr bằng 10, vòng lặp tự thoát. Vậy để nó in "1,2,3,4,5,6,7,8,9,10", chúng ta sẽ ghi range(1,11). Suy ra là vòng lặp sẽ kết thúc khi nó đạt đến chỉ số trên của giới hạn.

Ngoài ra, cần lưu ý đến cú pháp của câu lệnh. Đó là "for tên_biến in range(giá trị bắt đầu, giá trị cuối):". Dấu hai chấm ":" có nghĩa là chúng ta sắp bắt đầu một khối mã bên dưới. Điều quan trọng là không được quên dấu hai chấm ":" và thụt lề khối mã bên dưới cho đến khi nó kết thúc.

Nếu chúng ta thay đổi nội dung chương trình như sau:

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

for cntr in range(1,11):
        print cntr

print 'All Done'
```

Chúng ta sẽ nhận được...

```
greg@earth:~/python_examples$
./for_loop.py
1
2
3
4
5
6
7
8
9
10
All Done
greg@earth:~/python_examples$
```

Hãy đảm bảo rằng bạn thụt lề đúng. Luôn nhớ rằng sự thụt lề báo hiệu một khối mã lệnh. Chúng ta sẽ tìm hiểu nhiều hơn về thụt lề trong bài học tới.

Đó là những gì của bài học hôm nay. Lần tới chúng ta sẽ ôn lại và đi tiếp với nhiều câu lệnh lập trình python hơn. Trong lúc chờ đợi, các bạn có thể thử cài đặt một trình soạn thảo python như là Dr. Python hoặc SPE (Stani's Python Editor), cả hai đều có thể cài đặt thông qua Synaptic.

Greg Walters is owner of *RainyDay Solutions, LLC*, a consulting company in Aurora, Colorado, and has been programming since 1972. He enjoys cooking, hiking, music, and spending time with his family.

LÀM THẾ NÀO Lập trình Python – Phần 2

Viết bởi Greg Walters

Hiệu đính phần 1
David Turner đã gửi thư cho tôi nói về việc sử dụng Tab để thụt lề cho mã nguồn chương trình sẽ gây một chút lầm lẫn cho những trình soạn thảo sử dụng nhiều hoặc ít hơn 4 khoảng trắng một đoạn thụt lề. Điều này đúng. Nhiều lập trình viên Python (có cả tôi) tiết kiệm thời gian bằng cách đặt một tab trong trình soạn thảo của họ là bốn khoảng trắng. Vấn đề ở đây là có những người sử dụng những trình soạn thảo mà cài đặt khác bạn, nó có thể làm cho đoạn mã nguồn nhìn không được đẹp và có thể gặp một số rắc rối. Vì vậy hãy tập thói quen sử dụng khoảng trắng thay vì sử dụng Tab.

Trong bài lần trước, chúng ta đã được thấy một chương trình đơn giản sử dụng raw_input để lấy sự tương tác từ người dùng, một số kiểu biến đơn giản, và một vòng lặp đơn giản sử dụng biểu thức "for". Ở bài này chúng ta sẽ tập trung nhiều hơn vào các biến, và thử viết vài chương trình.

Danh sách

Giờ chúng ta sẽ tìm hiểu một kiểu biến khác đó là kiểu danh sách. Ở những ngôn ngữ khác, một danh sách có thể được xét như một mảng. Hãy liên tưởng tới những cái hộp đựng giầy, một mảng (hay danh sách) sẽ là một số lượng các hộp được dán lại cạnh nhau và ở mỗi hộp đều chứa một đồ vật. Ví dụ, chúng ta có thể đặt dĩa trong một cái hộp, dao ở cái hộp khác, và thìa ở một cái khác. Hãy xem một danh sách đơn giản. Một cách dễ dàng để hình dung là danh sách các tháng trong năm. Chúng ta có đoạn mã như thế này:

```
months =
['Jan','Feb','Mar','Apr','May
','Jun','Jul','Aug','Sep','Oc
t','Nov','Dec']
```

Để tạo một danh sách, chúng ta chúng ta nhóm tất cả các giá trị vào trong một cặp ngoặc vuông ('[' và ']'). Chúng ta đặt tên danh sách là 'months'. Để sử dụng nó, chúng ta có thể gọi kiểu như: print months[0] hoặc months[1] (nó sẽ in ra 'Jan' hoặc 'Feb'). Nhớ slaf chúng ta luôn luôn đếm từ 0. Để tìm độ dài của danh sách, chúng ta có thể sử dụng:

```
print len(months)
```

nó sẽ trả về 12.

Một ví dụ khác về danh sách là danh mục các sách nấu ăn. Ví dụ:

```
catagories = ['Main dish',
'Meat', 'Fish', 'Soup',
'Cookies']
```

Như trên thì categories[0] sẽ là 'Main dish', và categories[4] sẽ là 'Cookies'. Rất đơn giản. Tôi nghĩ rằng bạn có thể nghĩ ra rất nhiều thứ có thể sử dụng danh sách.

Đến giờ chúng ta đã tạo một danh sách sử dụng các xâu để chưa thông tin. Chúng ta cũng có thể tạo một danh sách sử dụng các số nguyên. Nhìn lại danh sách months, chúng ta có thể tạo một danh sách chứa số ngày của mỗi tháng:

```
DaysInMonth =
[31,28,31,30,31,30,31,31,30,3
1,30,31]
```

Nếu in ra DaysInMonth[1] (cho tháng 2) chúng ta sẽ được 28, nó là một số nguyên. Để ý rằng tôi lấy tên của danh sách là DaysInMonth. Không khó khăn gì nếu tôi lấy nó là 'daysinmonth' hoặc đơn giản chỉ là 'X'... nhưng nó hơi khó đọc. Những lập trình viên giỏi luôn yêu cầu các tên của biến phải dễ hiểu. Chúng ta sẽ đi vào chi tiết về vấn đề này sau. Chút nữa chúng ta sẽ nói thêm về danh sách.

Trước khi chúng ta đến với chương trình ví dụ tiếp theo, hãy xem một vài thứ khác về Python.

Nói thêm về Xâu

Chúng ta đã nói qua về xâu ở phần 1. Bây giờ đi sâu hơn một chút về chuỗi. Một xâu là một danh sách các kí tự. Không còn cách giải thích nào ngắn gọn hơn. Thực tế, bạn có thể coi một xâu như là một mảng các kí tự. Ví dụ nếu chúng ta

gắn xâu 'The time has come' vào biến tên là strng, và sau đó một biết kí tự thứ 2 của xâu là gì, chúng ta thử gõ:

```
strng = 'The time has come'
print strng[1]
```

Kết quả sẽ là 'h'. Nhớ rằng chúng ta luôn luôn đếm từ 0, vì vậy kí tự đầu tiên sẽ là [0], thứ hai là [1], thứ 3 sẽ là [2], và cứ thế. Nếu chúng ta muốn tìm những kí tự bắt đầu từ vị trí 4 đến vị trí 8, chúng ta gõ:

```
print strng[4:8]
```

nó sẽ trả về 'time'. Như vòng lặp for ở phần một, bộ đếm dừng lại ở 8, nhưng không trả về kí tự thứ 8, đó là khoảng trắng đằng sau 'time'.

Chúng ta có thể tìm độ dài của xâu bằng cách sử dụng hàm len():

```
print len(strng)
```

nó sẽ trả về 17. Nếu chúng ta muốn tìm từ 'time' nằm ở vị trí nào của xâu, có thể sử dụng

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

Bây giờ, biến pos (viết gọn của position) là 4, cho thấy từ 'time' bắt đầu ở vị trí 4 của xâu. Nếu chúng ta dùng hàm find để tìm một từ hoặc một chuỗi không có trong xâu như sau:

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

sẽ trả về giá trị -1 cho biến pos.

Chúng ta cũng có thể lấy ra các từ riêng lẻ trong xâu bằng cách sử dụng lệnh split. Chúng ta sẽ cắt (hoặc bẻ gãy) xâu tại mỗi khoảng trắng sử dụng:

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

nó sẽ trả về một danh sách chứa ['The', 'time', 'has', 'come']. Đây là một hàm rất hữu ích. Ngoài ra còn có nhiều hàm khác thao tác với xâu, chúng ta sẽ sử dụng chúng sau này.

Thay thế chuỗi

Có một việc nữa mà tôi muốn nói đến trước khi chúng ta bước vào chương trình ví dụ tiếp theo. Khi chúng ta muốn in ra cái gì đó bao gồm những đoạn văn bản và cả các biến, chúng ta có thể sử dụng Thay thế biến. Làm việc này khá đơn giản. Nếu chúng ta muốn thay thế một xâu, chúng ta sử dụng '%s' và chỉ định vị trí sẽ thay thế. Ví dụ, để in ra một tháng từ danh sách bên trên, chúng ta có thể sử dụng:

```
print 'Month = %s' %kmonth[0]
```

Nó sẽ in ra 'Month = Jan'. Nếu bạn muốn thay thế một số nguyên, sử dụng '%d'. Hãy xem ví dụ bên dưới:

```
Months = ['Jan', 'Feb',
'Mar', 'Apr', 'May', 'Jun',
'Jul', 'Aug', 'Sep', 'Oct',
'Nov', 'Dec']
DaysInMonth =
[31,28,31,30,31,30,31,31,30,3
1,30,31]
for cntr in range(0,12):
    print '%s has %d days.'
%(Months[cntr],
DaysInMonth[cntr])
```

Kết quả của đoạn mã trên sẽ là:

```
Jan has 31 days.
Feb has 28 days.
Mar has 31 days.
Apr has 30 days.
May has 31 days.
Jun has 30 days.
Jul has 31 days.
Aug has 31 days.
Sep has 30 days.
Oct has 31 days.
Nov has 30 days.
Dec has 31 days.
```

Cái quan trọng cần hiểu ở đây là sử dụng nháy đơn và nháy kép. Nếu bạn đặt một biến kiểu xâu như sau:

```
st = 'The time has come'
```

hoặc như sau:

```
st = “The time has come”
```

kết quả là tương tự nhau. Tuy vậy, nếu bạn cần thêm một dấu nháy đơn trong xâu như sau:

```
st = 'He said he's on his
way'
```

bạn sẽ gặp phải lỗi cú pháp. Bạn cần phải khởi tạo nó như sau:

```
st = “He said he's on his
way”
```

Nghĩ theo cách này. Để định nghĩa một xâu, bạn phải đặt nó vào trong một cặp dấu nháy (đơn hoặc kép) – một ở đầu và một ở cuối. Nếu bạn cần dùng nhiều loại dấu nháy, sử dụng cặp dấu nháy ngoài cùng là loại mà không xuất hiện trong xâu

như ở ví dụ trên. Nhưng bạn muốn hỏi, làm thế nào để khai báo một xâu dạng “She said “Don't Worry””? Trong trường hợp này, bạn có thể định nghĩa nó như sau:

```
st = 'She said “Don\'t
Worry”'
```

Chú ý dấu xổ chéo trước dấu nháy đơn trong 'Don't'. Nó được gọi là kí tự thoát, và báo cho Python biết rằng in ra một dấu nháy đơn chứ không phải là một dấu định nghĩa xâu. Một vài kiểu dùng kí tự thoát khác như '\n' cho dòng mới, và '\t' cho một tab. Chúng ta sẽ dùng chúng ở những ví dụ sau này.

Phép gán và biểu thức bằng nhau

Chúng ta cần học thêm vài thứ để có thể làm ví dụ tiếp theo. Đầu tiên là sự khác nhau giữa phép gán và 2 biểu thức bằng nhau. Chúng ta đã sử dụng phép gán nhiều lần trong những ví dụ. Khi chúng ta muốn đặt một giá trị cho một biến, chúng ta sử dụng phép gán, chúng ta sử dụng toán tử gán '=' (dấu bằng):

```
variable = value
```

Tuy vậy, khi chúng ta muốn ước lượng một biến và một giá trị, chúng ta phải sử dụng toán tử so sánh. Khi chúng ta muốn kiểm tra xem một biến có bằng một giá trị xác định nào đó không, chúng ta sử dụng '==' (2 dấu bằng):

```
variable == value
```

Vì vậy, nếu chúng ta có một biến tên là looop và chúng ta muốn biểu diễn nếu nó bằng 12, sử dụng:

```
if loop == 12:
```

Đừng lo lắng về if và dấu hai chấm ở ví dụ trên. Chỉ cần nhớ rằng chúng ta phải sử dụng 2 dấu bằng để ước lượng.

Chú thích

Điều tiếp theo chúng ta cần bàn tới là chú thích. Chú thích quan trọng cho nhiều việc. Không chỉ cho bạn hoặc ai đó thấy ý tưởng bạn đang làm, khi bạn quay lại với đoạn mã của mình sau 6 tháng, bạn có thể nhớ rõ bạn đang muốn làm gì. Khi bạn bắt đầu viết nhiều chương trình, chú thích sẽ trở nên rất quan trọng. Chú thích cũng cho phép bạn làm Python bỏ qua những dòng mã này. Để chú thích một dòng bạn sử dụng dấu '#'. Ví dụ:

```
# This is a comment
```

Bạn có thể đặt chú thích bất cứ đâu trên một dòng mã, nhưng nhớ rằng, khi đó Python sẽ bỏ qua những thứ sau dấu '#'.

Biểu thức if

Bây giờ chúng ta sẽ trở lại với biểu thức if trong ví dụ ngắn bên trên. Khi chúng ta muốn làm một việc dựa trên giá trị của cái gì đó, chúng ta sử dụng biểu thức if:

```
if loop == 12:
```

Nó sẽ kiểm tra biến 'loop', và, nếu giá trị là 12, thì sẽ làm mọi thứ ở khối thụt lề bên dưới. Nhiều trường hợp, như vậy là đủ, nhưng, khi bạn muốn dùng, nếu một biến bằng gì đó, thì làm cái này, không thì làm cái kia. Chúng ta có thể hình dung bằng đoạn mã giả sau:

```
if x == y then
      do something
else
      do something
```

và trong Python chúng ta có thể viết:

```
if x == y:
      do something
else:
      do something else
      more things to do
```

Những ý chính cần nhớ là:

2. Kết thúc biểu thức if hoặc else là dấu hai chấm.

1. Thụt lề các dòng mã của bạn

Hãy tự mình kiểm tra, bạn có thể sử dụng if/elif/else. Ví dụ:

```
x = 5
if x == 1:
      print 'X is 1'
elif x < 6:
      print 'X is less than
6'
elif x <10:
      print 'X is less than
10'
else:
      print 'X is 10 or
greater'
```

Chú ý rằng chúng ta sử dụng toán tử '<' để xác định nếu x NHỎ HƠN một gái trị nào đó –

trong trường hợp này là 6 hoặc 10. Những biểu thức so sánh thường dùng khác là lớn hơn '>', nhỏ hơn hoặc bằng '<=', lớn hơn hoặc bằng '>=', và không bằng '!='.

Biểu thức while

Cuối cùng, chúng ta sẽ xem một ví vụ đơn giản của biểu thức wile. Biểu thức while cho phép bạn tạo một vòng lặp làm hàng loạt những việc, cứ như vậy cho đến khi đặt tới một giới hạn xác định nào đó. Một ví dụ đơn giản là gắn một biến "loop" là 1. Sau đó khi biến loop có giá trị nhỏ hơn hoặc bằng 10 thì in giá trị của loop, thêm 1 vào loop và tiếp tục, đến khi loop lớn hơn 10 thì dừng lại:

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

chạy trong cửa sổ dòng lệnh chúng ta sẽ được như sau:

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

Đây chính là điều chúng ta muốn thấy. Minh họa 1 (ở trên, phải) là một ví dụ tương tự, nhìn có vẻ rắc rối hơn, nhưng vẫn đơn giản.

Trong ví dụ này, chúng ta kết hợp biểu thức if, vòng lặp while, biểu thức đầu vào, kí tự thoát thể hiện dòng mới, toán tử gán, và các toán tử so sánh – tất cả trong chương trình có 8 dòng.

Chạy ví dụ này sẽ được

```
Enter something or 'quit' to
end
 => FROG
You typed FROG
Enter something or 'quit' to
end
 => bird
You typed bird
Enter something or 'quit' to
end
 => 42
You typed 42
Enter something or 'quit' to
end
 => QUIT
You typed QUIT
Enter something or 'quit' to
end
 => quit
quitting
```

```
loop = 1
while loop == 1:
      response = raw_input("Enter something or 'quit' to
end => ")
      if response == 'quit':
            print 'quitting'
            loop = 0
      else:
            print 'You typed %s' % response
```

MH. 1

Chú ý rằng khi chúng ta gõ 'QUIT', chương trình không dùng. Bởi vì chúng ta ước lượng giá trị của biến trả về là 'quit' (response == 'quit'). 'QUIT' không bằng 'quit'.

Trước khi kết thúc tháng này, chúng ta tới với một ví dụ ngắn khác. Hãy viết bạn muốn kiểm tra xem nếu một người dùng được phép truy cập chương trình. Ví dụ này không phải là cách tốt nhất để làm việc này, nhưng nó tốt để thực hành với những gì chúng ta đã học.

Đơn giản, chúng ta sẽ hỏi người dùng tên và mật khẩu, so sánh chúng với thông tin đã viết bên trong chương trình, và sau đó thực hiện quyết định dựa trên những gì chúng ta thấy. Chúng ta sẽ sử dụng hai danh sách – một để chứa những người dùng được phép và một để chứa mật khẩu. Sau đó chúng ta sẽ sử dụng raw_input để lấy thông tin từ người dùng, và cuối cùng sử dụng if/elif/else để kiểm tra và quyết định nếu người dùng đó được phép. Nhớ rằng, đấy không phải là cách tốt nhất để làm điều này. Chúng ta sẽ đi vào những cách khác ở các bài sau. Đoạn mã của chúng ta ở ô bên phải.

Lưu lại thành 'password_test.py' và chạy với nhiều đầu vào khác nhau.

Chỉ còn một thứ chúng ta chưa nói đến là việc kiểm tra danh sách ở đoạn mã bắt đầu với 'if username in users:'. Chúng ta sẽ làm gì để kiểm tra xem nếu tên người dùng được nhập vào có trong danh sách. Nếu nó có, chúng ta sẽ lấy được vị trí của tên người dùng trong danh sách người dùng. Sau đó chúng ta sử dụng users.index(usrname) để lấy vị

trí trong danh sách người dùng như vậy chúng ta có thể lấy mật khẩu, nó được đặt cùng vị trí ở danh sách mật khẩu. Ví dụ, John là ở vị trí 1 trong danh sách người dùng. Mật khẩu của anh ấy, 'dog' là ở vị trí 1 trong danh sách mật khẩu. Theo cách này chúng ta có thể lấy ra các cặp.

Vậy là đủ cho tháng này. Tháng tới, chúng ta sẽ học về các hàm và mo-đun. Trong khi chờ đợi, hãy thực hành với những gì bạn đã học được và chúc vui vẻ.

```
#-----------------------------------------------
#password_test.py
#    example of if/else, lists, assignments,raw_input,
#    comments and evaluations
#-----------------------------------------------
# Assign the users and passwords
users = ['Fred','John','Steve','Ann','Mary']
passwords = ['access','dog','12345','kids','qwerty']
#-----------------------------------------------
# Get username and password
usrname = raw_input('Enter your username => ')
pwd  = raw_input('Enter your password => ')
#-----------------------------------------------
# Check to see if user is in the list
if usrname in users:
     position = users.index(usrname) #Get the position in the list of the users
     if pwd == passwords[position]:  #Find the password at position
          print 'Hi there, %s.  Access granted.' % usrname
     else:
          print 'Password incorrect.  Access denied.'
else:
     print "Sorry...I don't recognize you.  Access denied."
```

Greg Walters là chủ của RainyDay Solutions, LLC, một công ty tư vấn ở Aurora, Colorado, là lập trình viên từ năm 1972. Ông thích nấu nướng, đi bộ đường dài, nghe nhạc và dành thời gian cho gia đình.

LÀM THẾ NÀO Lập trình Python – Phần 3

Viết bởi Greg Walters

Trong bài viết trước, chúng ta đã học về danh sách, kí tự thay thế, chú thích, phép gán và biểu thức bằng nhau, câu lệnh if và while. Tôi đã hứa với các bạn là trong bài viết này chúng ta sẽ được học về mô-đun và hàm. Vậy thì chúng ta hãy bắt đầu.

Mô-đun

Mô-đun là một cách để có thể mở rộng việc lập trình Python của bạn. Bạn có thể tự tạo ra mô-đun riêng của mình hoặc sử dụng những cái sẵn có của Python, hoặc sử dụng những cái người khác đã tạo ra. Bản thân Python có sẵn hàng trăm mô-đun tạo thuận lợi cho việc lập trình của bạn. Danh sách các mô-đun sẵn có của Python có thể tìm thấy tại http://docs.python.org/modindex.html. Một vài mô-đun phụ thuộc vào đặc trưng của hệ điều hành, nhưng hầu hết đều hỗ trợ đa nền tảng (có thể được sử dụng trên Linux, Mac và Microsoft Windows). Để có thể sử dụng một mô-đun ngoài, bạn phải nạp mô-đun đó vào chương trình của mình. Một trong những mô-đun sẵn có của Python tên là 'random'. Mô-đun này cho phép bạn tạo những số (giả) ngẫu nhiên. Chúng ta sẽ sử dụng mô-đun này trong ví dụ đầu tiên:

```
#=====================================
# random_example.py
# Module example using the random module
#=====================================
import random
# print 14 random integers
for cntr in range(1,15):
      print random.randint(1,10)
```

Hãy phân tích từng dòng lệnh. Bốn dòng đầu tiên là chú thích. Chúng ta đã nói về chúng trong bài trước. Dòng thứ năm thông báo cho Python sử dụng mô-đun random. Chúng ta bắt buộc phải khai báo một cách tường minh cho Python để nạp.

Dòng thứ bảy tạo một vòng lặp 'for' để hiển thị 14 số ngẫu nhiên. Dòng thứ tám sử dụng hàm randint() để xuất ra một số nguyên ngẫu nhiên giữa 1 và 10. Chú ý rằng chúng ta phải nói cho Python biết hàm được gọi từ mô-đun nào. Chúng ta làm việc đó (trong trường hợp này) bằng cách viết random.randint. Vì sao chúng ta phải tạo mô-đun? Hãy nghĩ là nếu tất cả các hàm đều được nạp thẳng vào Python, không những Python sẽ phình to lên và chậm chạp mà còn làm cho việc sửa lỗi trở nên khó khăn. Khi sử dụng mô-đun, chúng ta có thể chia nhỏ các câu lệnh vào các nhóm theo những mục đích sử dụng nhất định. Ví dụ: nếu bạn không cần sử dụng chức năng của cơ sở dữ liệu, bạn không cần phải biết là có một mô-đun cho SQLite. Mặc dù vậy, khi bạn cần nó thì nó luôn có sẵn. (Chúng ta sẽ sử dụng mô-đun cơ sở dữ liệu trong loạt bài viết này.)

Một khi bạn bắt đầu lập trình Python, có lẽ bạn sẽ viết những mô-đun cho riêng bạn để có thể tiện sử dụng lại mà không cần gõ lại những đoạn mã đó. Nếu bạn cần thay đổi cái gì trong cụm mã đó, bạn có thể làm mà không ảnh hưởng gì mấy tới chương trình chính của mình. Tuy nhiên có một vài hạn chế ở đây, chúng ta sẽ đề cập đến điều này sau. Bây giờ, khi chúng ta đã sử dụng câu lệnh 'import random' trước đó, chúng ta ra lệnh cho Python cho phép sử dụng tất cả các hàm có trong mô-đun random. Nếu chỉ cần sử dụng riêng hàm randint(), chúng ta có thể thay đổi câu lệnh nạp mô-đun thành:

```
from random import randint
```

Bây giờ khi gọi hàm, chúng ta không cần sử dụng phần 'random.'. Đoạn mã ở trên sẽ thay đổi thành:

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

Hàm

Vừa rồi, chúng ta đã nạp mô-đun random và sử dụng hàm randint(). Một hàm là một đoạn mã được viết để có thể được gọi, thường là nhiều lần, dễ dàng bảo trì và giúp chúng ta tránh khỏi việc phải gõ đi gõ lại một đoạn lệnh nào đó nhiều lần. Khi bạn phải viết cùng một đoạn mã nào đó nhiều lần trong chương trình của mình thì đoạn mã đó nên được viết thành một hàm. Hai ví dụ đơn giản sau đây sẽ nói lên sự tiện lợi của việc sử dụng hàm. Giả sử chúng ta cần lấy hai số, cộng chúng, nhân chúng và cuối cùng là trừ chúng với nhau, sau đó là hiện các số và kết quả mỗi lần. Khó khăn hơn, chúng ta phải làm việc đó 3 lần với 3 cặp số. Đoạn mã sau có thể là một ví dụ:

```
#silly example
print 'Adding the two numbers %d and %d = %d ' % (1,2,1+2)
print 'Multiplying the two numbers %d and %d = %d ' % (1,2,1*2)
print 'Subtracting the two numbers %d and %d = %d ' % (1,2,1-2)
print '\n'
print 'Adding the two numbers %d and %d = %d ' % (1,4,1+4)
print 'Multiplying the two numbers %d and %d = %d ' % (1,4,1*4)
print 'Subtracting the two numbers %d and %d = %d ' % (1,4,1-4)
print '\n'
print 'Adding the two numbers %d and %d = %d ' % (10,5,10+5)
print 'Multiplying the two numbers %d and %d = %d ' % (10,5,10*5)
print 'Subtracting the two numbers %d and %d = %d ' % (10,5,10-5)
print '\n'
```

Không những phải gõ rất nhiều, nó có thể dẫn tới lỗi, có thể khi gõ hay khi thay đổi một cái gì đó sau này. Thay vào đó, chúng ta sẽ tạo một hàm tên là 'DoTwo' sẽ nhận hai số và tính toán, hiển thị kết quả mỗi lần. Chúng ta bắt đầu bằng cách sử dụng từ khoá 'def' (ý chỉ rằng chúng ta đang định nghĩa một hàm). Sau 'def', chúng ta viết tên của hàm và một danh sách các tham số (nếu có) trong ngoặc đơn. Dòng định nghĩa kết thúc bởi một dấu hai chấm (:). Đoạn mã trong hàm phải được thụt lề. Và dưới đây là ví dụ thứ 2:

Như đã thấy, chúng ta chỉ phải gõ 8 dòng thay vì 12 dòng. Nếu chúng ta cần thay đổi gì đó trong hàm, chúng ta có thể làm việc đó mà không làm ảnh hưởng nhiều tới chương trình chính. Để gọi một hàm, chúng ta sử dụng tên hàm và sau đó là các tham số.

Sau đây là một ví dụ khác về hàm. Với những yêu cầu sau:

Chúng ta cần tạo một chương trình hiển thị một danh sách của những mặt hàng được định dạng một cách gọn gàng. Giống như hình ở trang sau.

Với giá của từng mặt hàng và giá tổng cộng được tính theo đồng đô-la và xu. Độ rộng của bảng có thể thay đổi được. Giá trị bên trái và bên phải cũng có thể thay đổi được. Chúng ta sử dụng 3 hàm để làm việc này. Một hàm hiển thị dòng đầu và dòng cuối, một hàm hiển thị các mặt hàng cũng như dòng tổng cộng và cuối cùng là một hàm để hiển thị dòng phân cách. May mắn là Python có thể giải quyết nhiều vấn đề một cách dễ dàng. Nếu bạn nhớ lại, chúng ta đã từng xuất ra một chuỗi nhân với 4 và nhận được bốn bản sao của chuỗi đó. Chúng ta có thể sử dụng điều đó trong trường hợp này. Để

```
#silly example 2...still silly, but better
def DoTwo(num1,num2):
    print 'Adding the two numbers %d and %d = %d ' % (num1,num2,num1+num2)
    print 'Multiplying the two numbers %d and %d = %d ' % (num1,num2,num1*num2)
    print 'Subtracting the two numbers %d and %d = %d ' % (num1,num2,num1-num2)
    print '\n'

DoTwo(1,2)
DoTwo(1,4)
DoTwo(10,5)
```

hiển thị dòng đầu hoặc cuối, chúng ta có thể lấy độ rộng, trừ đi 2 cho hai dấu '+' và sử dụng "'=' * (width -2)" (width là độ rộng của bảng). Để đơn giản hơn, chúng ta sử dụng thay thế biến để cho tất cả phần tử vào một dòng. Dòng lệnh sẽ có dạng : '%s%s%s' % ('+', ('=' * (width-2)), '+'). Giờ chúng ta có thể cho hàm xuất ra trực tiếp nhưng chúng ta sẽ sử dụng từ khoá "return" để trả chuỗi được tạo về dòng lệnh gọi hàm. Chúng ta đặt tên hàm là 'TopOrBottom' và đoạn mã cho hàm sẽ là:

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

Chúng ta có thể bỏ chú thích nhưng sẽ tốt hơn là chỉ ra tham số 'width' là gì. Để gọi hàm, chúng ta viết 'print TopOrBottom(40)' hoặc thay 40 bằng một độ rộng khác. Như vậy đã có một hàm để hiển thị hai dòng cần thiết. Chúng ta có thể tạo một hàm khác cho chuỗi phân cách sử dụng cách viết tương tự... HOẶC, chúng ta có thể thay đổi hàm trên bằng cách thêm vào tham số chỉ ra kí tự nằm giữa các dấu cộng. Thử làm điều đó và vẫn gọi hàm là TopOrBottom.

```
def
TopOrBottom(character,width):
     # width is total width
of returned line
     # character is the
character to be placed
between the '+' characters
     return '%s%s%s' %
('+',(character * (width-
2)),'+')
```

Giờ chúng ta có thể thấy sự cần thiết của chú thích. Nhớ rằng chúng ta đang trả lại chuỗi tạo được do vậy cần phải có cái gì đó để nhận nó khi gọi hàm. Thay vì gán vào một chuỗi khác, chúng ta sẽ hiển thị nó ra luôn. Đây là dòng gọi hàm.

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

Chúng ta không những giải quyết được ba dòng mà chúng ta còn rút ngắn được số hàm cần dùng từ 3 xuống 2. Chỉ còn lại mỗi phần giữa của bảng nữa.

Hãy gọi hàm tiếp theo là 'Fmt'. Chúng ta sẽ truyền cho nó bốn tham số:

val1 - Giá trị bên trái
leftbit - Độ rộng của "cột" bên trái
val2 - Giá trị bên phải (là một số dấu chấm động)
rightbit - Độ rộng của "cột" bên phải

Việc đầu tiên phải làm là định dạng dữ liệu ở phía bên phải. Do phải định dạng dữ liệu để hiển thị theo đô-la và xu, chúng ta có thể dùng một hàm đặc biệt (cho sự thay thế biến) để hiển thị giá trị dấu chấm động với n số thập phân sau dấu phân cách. Câu lệnh sẽ là '%2.f'. Chúng ta sẽ gán nó cho biến 'part2'. Câu lệnh đầy đủ sẽ là: 'part2 = '%2.f' % val2'. Ta cũng có thể sử dụng một số hàm mặc định của Python để xử lí chuỗi là ljust và rjust. Ljust sẽ canh trái chuỗi, thêm vào phía bên phải với một kí tự nào đó bạn chỉ định. Rjust cũng hoạt động như vậy nhưng phần thêm vào sẽ ở phía bên trái. Giờ thử xem việc sử dụng cách trộn biến vào một chuỗi lớn và trả về cho phần câu lệnh gọi. Sau đây là dòng tiếp theo.

```
return '%s%s%s%s' % ('| ',
val1.ljust(leftbit - 2, ' '),
part2.rjust(rightbit - 2, '
'), ' |')
```

Mới nhìn vào thì có vẻ trông rất là phức tạp, nhưng thử phân tích từng phần để thấy nó rất đơn giản:

Return - Chúng ta gửi chuỗi tạo được về cho câu lệnh gọi.
'%s%s%s%s' - Chúng ta sẽ ghép 4 giá trị vào chuỗi, mỗi giá trị sử dụng một %s làm phần giữ chỗ.
% (- Bắt đầu danh sách biến
'| ', - Print these literals
val1.ljust((leftbit - 2), ' ') - Nhận biến val1, canh lề trái với dấu cách cho (leftbit - 2) ký tự. Chúng ta trừ đi 2 cho hai kí tự '| ' ở phía bên trái.
part2.rjust(rightbit - 2, ' ') - Canh lề phải chuỗi với rightbit - 2 dấu cách.
' |' - kết thúc chuỗi.

Đó là tất cả những gì trong câu lệnh trên. Trong khi kiểm tra lỗi, bạn có thể thử nghiệm

```
'+==============================+'
'| Item 1                X.XX |'
'| Item 2                X.XX |'
'|------------------------------|'
'| Total                 X.XX |'
'+==============================+'
```

với nó. Thực sự, hàm Fmt chỉ có hai dòng ngoài dòng dịch nghĩa và chú thích. Chúng ta có thể gọi nó như sau.

```
print Fmt('Item
1',30,item1,10)
```

Cũng như trên chúng ta có thể gán giá trị trả lại vào một chuỗi nhưng chúng ta chỉ hiển thị nó ra. Ở câu lệnh trên chúng ta gửi 30 cho độ rộng bên trái và 10 cho độ rộng bên phải. Tổng cộng sẽ bằng giá trị 40 chúng ta gửi cho TopOrBottom trước đó. Giờ hãy bật trình soạn thảo và gõ vào đoạn mã sau.

Lưu đoạn mã thành tập tin 'pprint1.py' và chạy nó. Bạn sẽ nhận được.

Mặc dù đây chỉ là một thí dụ đơn giản nhưng từ đó bạn có thể hiểu tại sao và làm thế nào sử dụng hàm. Giờ thử mở rộng một chút và học thêm về danh sách. Bạn còn nhớ về phần 2 khi chúng ta nói về danh sách? Có một điều mà tôi đã không nói với các bạn là một danh sách có thể chứa mọi thứ, kể cả những danh sách. Thử định nghĩa một danh sách mới trong chương trình với tên là 'itms' như sau:

```
itms =
[['Soda',1.45],['Candy',.75],
['Bread',1.95],['Milk',2.59]]
```

Nếu chúng ta muốn truy xuất một danh sách bình thường, chúng ta sẽ dùng print itms[0]. Tuy vậy, cái chúng ta nhận được sẽ là ['Soda', 1.45], không phải là thứ chúng ta cần trong trường hợp này. Chúng ta muốn truy xuất tất cả các phần tử trong danh sách đầu tiên. Chúng ta sẽ dùng 'print itms[0][0]' để có 'Soda' và [0][1] để có phần giá bán, chính là 1.45. Giờ chúng ta có 4 mặt hàng đã mua và chúng ta muốn sử dụng thông tin đó để hiển thị ra. Chúng ta chỉ cần thay đổi phần cuối của chương trình. Lưu lại chương trình trước đó thành 'pprint2.py', rồi ghi chú hai dòng định nghĩa itemx và thêm vào danh sách mà chúng ta vừa

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

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

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

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

đề cập. Đoạn mã sẽ thành.

```
#item1 = 3.00
#item2 = 15.00
itms =
[['Soda',1.45],['Candy',.75],
['Bread',1.95],['Milk',2.59]]
```

Tiếp theo, xoá bỏ hết các dòng lệnh gọi Fmt(). Sau đó thêm vào các dòng sau (những cái có chú thích #NEW LINE ở cuối) để cho đoạn mã của bạn trông giống như.

Tôi đặt một biến đếm cho vòng lặp sẽ chạy suốt danh sách. Nhớ rằng tôi vừa thêm vào một biến tên là total. Chúng ta gán cho biến total là 0 trước khi sử dụng vòng lặp. Khi hiển thị từng mặt hàng đã bán, chúng ta cộng thêm giá vào total. Cuối cùng, chúng ta hiển thị total ngay sau dòng phân cách. Lưu chương trình lại và chạy nó. Bạn sẽ nhận được kết quả tương tự như hình phía dưới.

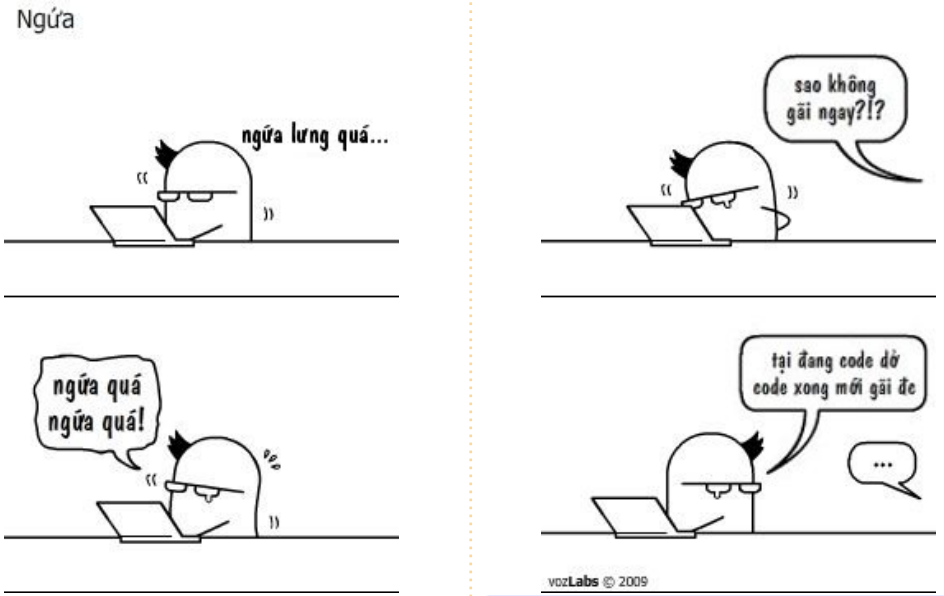
Nếu bạn muốn hơn nữa, bạn có thể thêm một dòng cho phần thuế. Giống như đã xử lý dòng total nhưng thay vào đó sử dụng (total * .086) thay cho phần giá.

```
print
Fmt('Tax:',30,total*
.086,10)
```

Bạn cũng có thể thêm nhiều phần tử vào danh sách và xem chương trình chạy thế nào.

Đó là tất cả những gì tôi muốn nói trong lần này. Lần tới, chúng ta sẽ tập trung vào lớp của Python. **Chúc vui!**

```
itms = [['Soda',1.45],['Candy',.75],['Bread',1.95],['Milk',2.59]]

print TopOrBottom('=',40)

total = 0 #NEW LINE
for cntr in range(0,4): #NEW LINE
      print Fmt(itms[cntr][0],30,itms[cntr][1],10) #NEW LINE
      total += itms[cntr][1] #NEW LINE
print TopOrBottom('-',40)
print Fmt('Total',30,total,10) #CHANGED LINE
print TopOrBottom('=',40)
```




```
+======================================+
| Soda                          1.45   |
| Candy                         0.75   |
| Bread                         1.95   |
| Milk                          2.59   |
+--------------------------------------+
| Total                         6.74   |
+======================================+
```

Greg Walters là chủ của RainyDay Solutions, LLC, một công ty tư vấn ở Aurora, Colorado, là lập trình viên từ năm 1972. Ông thích nấu nướng, đi bộ đường dài, nghe nhạc và dành thời gian cho gia đình.

LÀM THẾ NÀO Lập trình Python – Phần 4

Viết bởi Greg Walters

Lần trước tôi đã hứa với các bạn là kì này chúng ta sẽ thảo luận về lớp. Vì thế, chúng ta sẽ tập trung vào nó. Lớp là gì và nó có lợi ích như thế nào?

Lớp là một cách thức tạo nên đối tượng. Một đối tượng chỉ đơn giản là một cách thể hiện các thuộc tính và hành vi theo một nhóm. Điều này nghe có vẻ khó hiểu nhưng tôi sẽ giải thích cụ thể cho các bạn. Suy nghĩ thế này, một đối tượng là cách thức mô hình hóa cái gì đó trong thực tế. Lớp là phương pháp dùng để thực hiện điều đó. Ví dụ, chúng ta có ba con chó ở nhà: Beagle, Lab và một con lai Shepherd/Heeler xanh của Đức. Cả ba đều là chó nhưng chúng khác nhau. Chúng có những đặc điểm chung nhưng cũng có những điểm riêng biệt. Ví dụ, Beagle thì thấp, mập mạp, màu nâu và hung dữ. Lab thì vừa vừa, màu đen và rất hiền hòa. Con lai Shepherd/Heeler thì cao, ốm, màu đen và hơi ngốc nghếch. Dễ dàng nhận thấy là một số thuộc tính thì rõ ràng. Thấp/vừa vừa/cao là những thuộc tính về chiều cao. Hung dữ, hiền hòa và ngốc nghếch là những thuộc tính về tính tình. Trên khía cạnh hành vi, chúng ta có thể có: ăn, ngủ, chơi và nhiều hành động khác.

```
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
```

Cả ba con đều thuộc về lớp Dog (chó). Trở lại các thuộc tính chúng ta đã dùng để miêu tả phía trên, ta có các dạng như: Dog.Name (tên), Dog.Height (chiều cao), Dog.Build (ốm, mập, ...) và Dog.Color (màu). Chúng ta cũng có các hành vi như: Dog.Bark (sủa), Dog.Eat (ăn), Dog. Sleep (ngủ), ...

Như tôi đã đề cập, mỗi con chó thuộc mỗi giống khác nhau. Mỗi giống chó sẽ là một lớp con của lớp Dog. Mối quan hệ được biểu diễn như sơ đồ sau:

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

Mỗi lớp con thừa kế tất cả các thuộc tính của lớp Dog. Vì thế, nếu chúng ta tạo một thể hiện của Beagle, nó sẽ có tất cả các thuộc tính từ lớp cha Dog.

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

Mọi thứ đã bắt đầu rõ ràng hơn rồi đúng không? Hãy tạo một lớp Dog hoàn chỉnh (phía trên). Chúng ta sẽ bắt đầu với từ khóa “class” và tên của lớp.

Trước khi giải thích kĩ hơn vào đoạn mã, chú ý hàm chúng ta đã định nghĩa ở đây. Hàm __init__ (hai dấu gạch dưới + 'init' + hai dấu gạch dưới) là hàm khởi tạo đối với mọi lớp. Ngay khi lớp được gọi trong mã thì đoạn chương trình này sẽ được chạy. Trong trường hợp này, chúng ta phải thiết lập một số tham số để tạo một vài thông tin cơ bản về lớp của chúng ta, bao gồm: tên, màu, chiều cao, tầm vóc, tính tình, tuổi và một cặp biến Hungry (đói) và Tired (mệt). Chúng ta sẽ trở lại trong lát nữa. Bây giờ hãy thêm vào một số dòng mã.

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

Đây là những dòng mã không dự tính trước (sẽ làm) và nằm bên ngoài lớp của chúng ta. Đoạn mã sử dụng lớp Dog. Dòng đầu tiên tạo một thể hiện của lớp Dog tên là Beagle. Nó được gọi là một thực thể cụ thể. Khi viết những dòng mã trên, chúng ta cũng đã truyền vào một vài thông tin cho thực thể của lớp này, như là tên của Beagle, màu và nhiều thứ khác nữa. Bốn dòng tiếp theo đơn giản chỉ truy vấn thông tin từ đối tượng Beagle và lấy những thông tin trả về đó. Lúc này chúng ta cần viết thêm vài dòng mã nữa. Thêm vào những dòng mã trên góc bên phải của trang này vào lớp ngay sau hàm __init__.

Bây giờ chúng ta có thể gọi Beagle.Eat() và Beagle.Sleep(). Hãy thêm vào một phương thức nữa, Bark. Đoạn mã được trình bày bên phải đây.

Cái này tôi viết cho nó linh hoạt hơn. Tùy thuộc vào tính tình của con chó, tiếng sủa sẽ thay đổi. Toàn bộ đoạn mã đầy đủ được trình bày ở trang kế tiếp.

Khi chúng ta chạy thì sẽ được kết quả như sau:

```
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
```

Hiện tại, chúng ta đã có được con Beagle già hung dữ. Tuy nhiên, như tôi đã nói ngay từ đầu là có đến 3 con chó. Bởi vì chúng ta đã viết mã cho lớp Dog một cách cẩn thận rồi, tất cả những gì chúng ta cần làm là tạo thêm hai thể hiện nữa cho lớp.

```
Lab = Dog('Nina', 'Black',
'Medium', 'Heavy', 'Laid
Back', 7)
Heeler = Dog('Bear', 'Black',
'Tall', 'Skinny', '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
```

```
    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'
```

```
Lab.Bark()
Heeler.Bark()
```

Chú ý rằng tôi đã tạo ra các thể hiện cho cả hai con chó trước khi tôi thực hiện câu lệnh print. Điều này không làm nảy sinh vấn đề gì cả bởi vì tôi đã định nghĩa thể hiện trước khi gọi bất kì phương thức nào. Đây là toàn bộ những gì xuất ra màn hình của chương trình cho lớp Dog.

```
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
```

Giờ đây bạn đã có những kiến thức cơ bản nhất, bài tập về nhà dành cho bạn là mở rộng lớp Dog đã viết thêm nhiều phương thức hơn, ví dụ như là Play(chơi) hay EncounterStrangeDog(chạm trán một con chó lạ), đại loại như vậy.

Lần tới, chúng ta sẽ bắt đầu thảo luận về GUI hay còn gọi là lập trình giao diện người dùng đồ họa. Chúng ta sẽ sử dụng **Boa Constructor** để thực hiện.

```
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()
```

Greg Walters là chủ của RainyDay Solutions, LLC, một công ty tư vấn ở Aurora, Colorado, là lập trình viên từ năm 1972. Ông thích nấu nướng, đi bộ đường dài, nghe nhạc và dành thời gian cho gia đình.

LÀM THẾ NÀO Lập trình Python – Phần 5

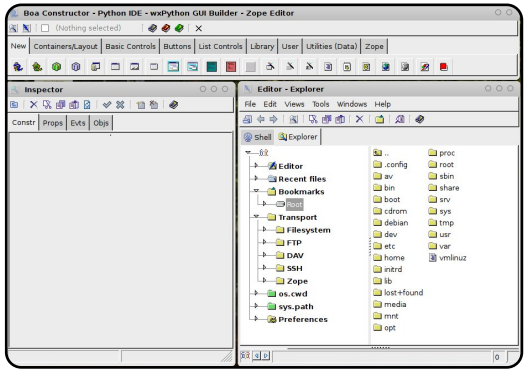
Viết bởi Greg Walters

Nếu bạn cũng như tôi, bạn sẽ ghét phần đầu tiên của việc cài đặt này. Tôi ghét nó khi tác giả nào đó nói với tôi rằng tôi phải đọc thật tỉ mỉ mỗi từ trong sách/chương/bài viết của họ, bởi vì tôi chỉ biết rằng nó sẽ là một thứ làm cho tôi ngái ngủ - thậm chí tôi biết nó tốt cho chính tôi, và cuối cùng tôi sẽ làm điều đó bằng bất cứ cách nào.

Hãy cân nhắc đến những gì bạn đã được cảnh báo. Hãy đọc những thứ nhàm chán sau đây thật cẩn thận. Chúng ta sẽ sớm tận hưởng những điều thú vị, nhưng chúng ta cần thiết lập một vài bước cơ sở trước khi có thể thực sự nói về việc lập trình.

Đầu tiên, bạn cần cài đặt Boa Constructor và wxPython. Dùng Synaptic và chọn cả hai wxPython, Boa Constructor. Một khi đã cài đặt, bạn có thể tìm Boa ở Application|Programming|Boa Constructor. Hãy tìm và khởi chạy nó đi. Một khi ứng dụng bắt đầu, bạn sẽ thấy ba cửa sổ khác nhau (hoặc các khung): một ở trên và hai ở phía dưới. Bạn có lẽ phải định dạng lại và di chuyển chúng một ít, sao cho trông giống như sau:

Khung trên cùng được gọi là khung công cụ. Khung phía dưới bên trái là khung kiểm tra, và khung bên phải là khung soạn thảo. Trên khung công cụ, chúng ta có một số thẻ tab (New, Containers/Layout, v.v..) chúng sẽ cho phép bạn tạo dự án mới, thêm các khung vào các dự án hiện có, và thêm các điều khiển khác nhau vào các khung cho ứng dụng của bạn. Khung kiểm tra sẽ trở nên rất quan trọng khi chúng ta thêm các điều khiển vào ứng dụng. Khung soạn thảo sẽ cho phép chúng ta thêm đoạn mã, lưu các dự án, và nhiều hơn thế. Trở lại với khung công cụ, hãy nhìn vào mỗi tab – bắt đầu với tab "New". Trong khi có quá nhiều tùy chọn ở đó, chúng ta sẽ chỉ nói về hai trong số chúng. Chúng là nút thứ 5 và 6 từ trái qua: wx.App và wx.Frame. wx.App cho phép chúng ta tạo ứng dụng hoàn chỉnh có sẵn hai tệp tin. Một là tệp tin khung và còn lại là tệp tin ứng dụng. Đây là phương pháp tôi thích dùng. wx.Frame được dùng để thêm nhiều khung vào ứng dụng của chúng ta và/hoặc tạo một ứng dụng độc lập từ một tệp tin nguồn đơn lẻ. Chúng ta sẽ bàn về điều này sau.



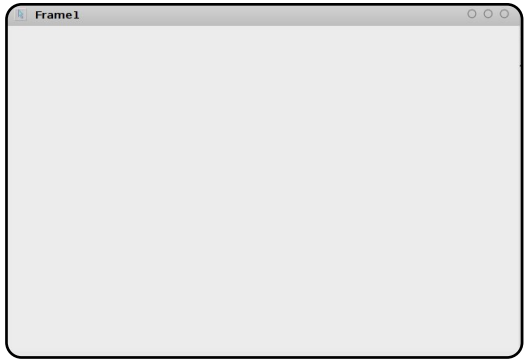
Bây giờ hãy nhìn vào thẻ Containers/Layout. Nhiều thứ thú vị ở đây. Một thứ bạn sẽ dùng nhiều là wx.Panel (đầu tiên bên trái) và các tùy chọn sắp xếp (vị trí 2, 3, 4, 5 và 6 từ phải qua). Trong thẻ Basic Controls bạn sẽ thấy những điều khiển như văn bản tĩnh (nhãn), các ô nhập liệu, các ô duyệt, các nút radio, và nhiều hơn. Dưới thẻ Buttons, bạn sẽ thấy các hình thức khác nhau của nút. Thẻ List có dữ liệu dạng bảng và các ô danh sách khác. Chuyển đến thẻ Utilities, chúng ta sẽ tìm thấy bộ đếm thời gian và bộ thiết kế trình đơn.

Đây là một số điều cần nhớ trước khi chúng ta sẵn sàng cho ứng dụng đầu tiên. Có một vài lỗi trong phiên bản Linux. Một là một vài điều khiển sẽ không cho phép bạn di chuyển chúng trong chế độ thiết kế. Dùng <Ctrl> + Phím mũi tên để di chuyển hoặc thay đổi vị trí các điều khiển của bạn. Một lỗi khác bạn sẽ tìm thấy khi thử làm những bài hướng dẫn với Boa Constructor, là khi đặt một bảng điều khiển, nó hơi khó để nhìn thấy. Hãy tìm các ô nhỏ (tôi sẽ chỉ chúng cho bạn sớm thôi). bạn có thể dùng thẻ Objs trên khung kiểm tra và chọn nó.

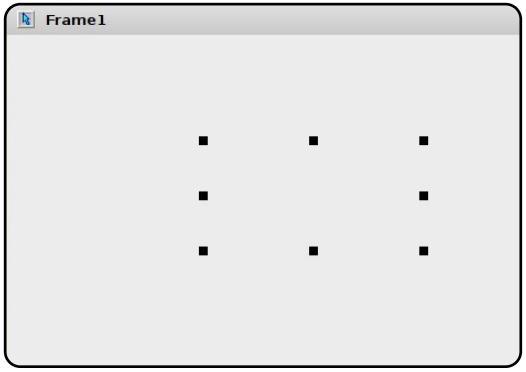
Được rồi, chúng ta đi tiếp. Phía dưới tab "New" của khung

công cụ, chọn wx.App (nút thứ 5 từ trái sang). Nó sẽ tạo hai tab mới trong khung soạn thảo: một được đặt tên là "*(App1)*", hai là "*(Frame1)*". Có vẻ điên rồ nhưng điều quan trọng đầu tiên chúng ta sẽ làm là lưu hai tệp tin mới này, bắt đầu với tệp Frame1. Nút lưu là nút thứ 5 từ trái qua trong khung soạn thảo. Khung "Save as" sẽ hiện một cửa sổ hỏi bạn nơi bạn muốn lưu tệp tin và bạn muốn đặt tên nó là gì. Tạo một thư mục trong thư mục chính của bạn tên là GuiTests, và lưu tệp với tên "Frame1.py". Chú ý tên tab "*(Frame1)*" giờ là "Frame1". (Ký hiệu "*(" nói rằng tệp tin đó cần được lưu.). Bây giờ làm tương tự với tab App1. Bây giờ hãy quan sát một số nút trên thanh công cụ của khung soạn thảo. Những cái quan trọng là sao lưu (phím thứ 5 từ trái qua) và thực thi (mũi tên màu vàng, thứ 7 từ trái qua. Nếu bạn đang trong một tab khung (ví dụ Frame1) bạn sẽ cần phải biết một vài nút phụ. Đó là nút Designer (thiết kế) 

Nó là một thứ quan trọng. Nó cho phép chúng ta thiết kế khung giao diện GUI – thứ mà chúng ta sẽ làm lúc này. Khi bạn nhấn vào nó bạn sẽ thấy một khung trống hiện lên.



Đây là một khung trống để bạn đặt bất cứ điều khiển nào bạn cần (một cách hợp lý). Điều đầu tiên chúng ta muốn làm là đặt điều khiển wx.panel. Hầu hết mọi thứ tôi từng đọc đều nói rằng không nên đặt các điều khiển (ngoại trừ wx.panel) trực tiếp vào trong khung. Vì vậy, nhấn vào tab Containers/Layout trong khung công cụ, rồi nhấn chọn wx.panel. Kế đến, chuyển đến khung mới hiện mà bạn đang thao tác và nhấn vào một chỗ nào đó trong khung. Nếu bạn thấy giống như thế này thì bạn đã làm đúng:

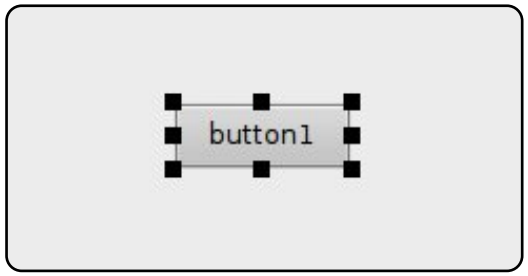


Còn nhớ khi tôi cảnh báo bạn về các lỗi chứ? Đây là một trong số chúng. Đừng quá lo lắng. Bạn thấy 8 ô vuông màu đen nhỏ chứ? Đó chính là giới hạn của panel. Nếu bạn muốn, bạn có thể nhấn và kéo một trong số chúng để thay đổi kích cỡ của panel, nhưng đối với dự án này điều chúng ta muốn là làm cho panel chiếm toàn bộ khung. Chỉ cần co kích cỡ của khung lại một chút. Bây giờ chúng ta có một cái bảng (panel) để gắn các điều khiển vào. Di chuyển khung bạn đang làm việc cho tới khi bạn có thể nhìn thấy thanh công cụ của khung soạn thảo. Hai nút mới đã xuất hiện: một nút duyệt (Check) và một nút "X". "X" sẽ hủy bỏ các thay đổi bạn vừa thực hiện.

Nút duyệt: 

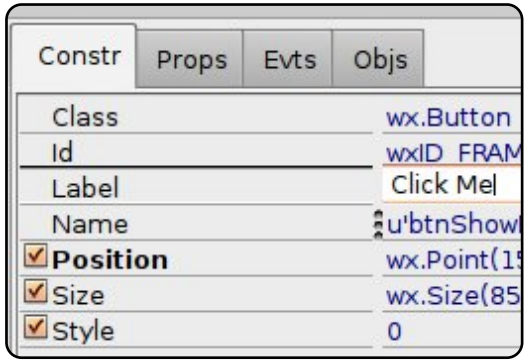
còn được gọi là nút "Post". Nó sẽ ghi các thay đổi vào trong tệp tin khung. Bạn vẫn phải lưu tệp tin khung đó, nhưng sẽ có một vài thứ mới trong tệp tin. Vì vậy, nhấn vào nút Post. Cũng có một nút post trong khung kiểm tra, nhưng chúng ta sẽ đề cập sau. Bây giờ hãy lưu tệp tin của bạn.

Trở lại chế độ thiết kế. Nhấn vào tab 'Buttons' trên khung công cụ và sau đó nhấn vào nút đầu tiên bên trái, wx.Button. Sau đó gắn nó vào một nơi nào đó gần giữa khung của bạn. Bạn sẽ thấy một thứ gần giống như thế này:

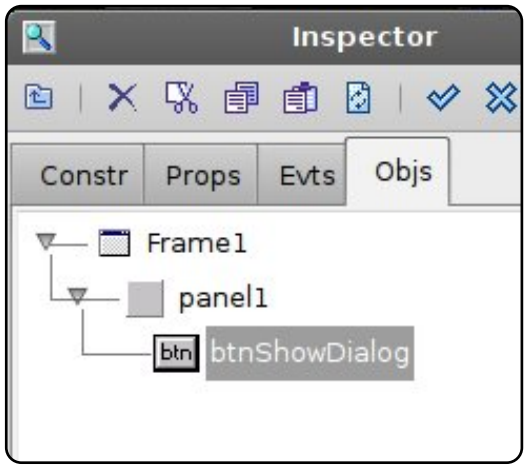


Chú ý rằng có 8 ô vuông nhỏ xung quanh nó giống như bảng panel. Chúng là các mốc thay đổi kích thước. Nó cũng cho chúng ta thấy điều khiển nào đang được chọn. Để di chuyển nút tới gần hơn phần trung tâm

của khung, giữ phím Control (Ctrl) và trong khi đang nhấn, dùng các phím mũi tên để di chuyển chúng tới nơi bạn muốn. bây giờ, hãy nhìn vào khung kiểm tra. Có bốn thẻ tab. Nhấn vào tab 'Constr'. Tại đây chúng ta có thể thay đổi nhãn, tên, vị trí, kích thước và kiểu dáng. Bây giờ, hãy thay đổi tên (Name) thành 'btnShowDialog' và nhãn (Label) thành 'Click Me'.



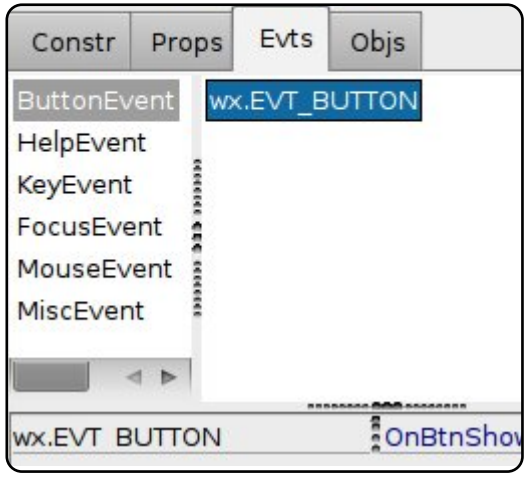
Bây giờ, hãy bỏ qua tất cả các tab còn lại và đến tab Objs. Tab này cho thấy tất cả các điều khiển bạn có và cấp bậc giữa chúng. Như bạn thấy, nút này là một con của panel1, panel1 là con của Frame1.



Bấm nút duyệt và lưu thay đổi của bạn. Trở lại thiết kế một lần nữa, và chú ý (giả sử bạn vẫn mở tab 'Objs' trong khung kiểm tra) rằng Frame1 hiện đang được chọn. Thật tiện, bởi vì chúng ta muốn thế. Trở lại tab 'Constr', và thay đổi tiêu đề (title) từ 'Frame1' thành 'Our First GUI'. Duyệt và lưu lại một lần nữa. Bây giờ hãy chạy ứng dụng của chúng ta. Nhấn vào nút Run màu vàng trên khung soạn thảo.

Nhấn vào nút bao nhiêu lần bạn muốn, nhưng sẽ chẳng có gì xảy ra. Tại sao? À, chúng ta đã không bảo nút đó làm bất cứ điều gì. Vì vậy, chúng ta cần thiết lập một sự kiện để kích hoạt khi người dùng nhấn vào nút của chúng ta. Nhấn vào chữ X góc ngoài cùng bên phải để đóng khung đang chạy. Kế tiếp, trở lại thiết kế, chọn nút và đến tab 'Evts' trong khung kiểm tra. Nhấn vào ButtonEvent và sau đó nhấp đôi vào dòng wx.EVT_BUTTON vừa xuất hiện, và để ý trong cửa sổ bên dưới chúng ta có một nút sự kiện được gọi là 'OnBtnShowDialogButton'. Duyệt và lưu lại.

Trước khi chúng ta tiến xa



hơn, hãy xem chúng ta có gì trong đoạn mã (trang tiếp theo).

Hàng đầu tiên là một chú thích bảo Boa Constructor rằng đây là một tệp tin boa. Nó được bỏ qua bởi chương trình biên dịch của Python, nhưng với Boa thì không. Dòng tiếp theo nhập thư viện wxPython. Bây giờ hãy chuyển tới định nghĩa lớp.

Ở trên cùng là phương thức __init_ctrls. Lưu ý chú thích ngay dưới dòng định nghĩa. Đừng chỉnh sửa mã trong phần này. Nếu bạn làm điều đó, bạn sẽ gặp rắc rối đấy. Bất cứ thứ gì trong phương thức này cần được bảo toàn. Trong đây, bạn sẽ tìm thấy các định nghĩa của mỗi điều khiển trong khung của chúng ta.

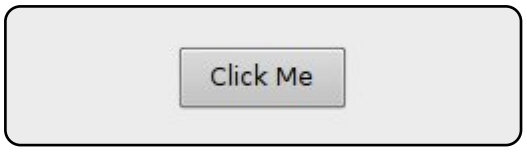
Kế đến, hãy chú ý hàm __init__. Tại đây, bạn có thể đặt những thứ liên quan tới mã khởi tạo. Cuối cùng là hàm OnBtnShowDialogButton. Đây là nơi bạn sẽ đặt mã kích hoạt khi người dùng nhấn nút. Lưu ý có một dòng event.Skip(). Nói một cách đơn giản, sự kiện sẽ kết thúc ngay khi nó vừa được kích hoạt.

Bây giờ, những gì chúng ta sẽ làm là gọi một hộp thông báo kèm theo một thông điệp. Đây là một việc làm bình thường đối với lập trình viên cho phép người dùng biết về một điều gì đó - một lỗi, hoặc

thông báo một quy trình đã hoàn thành. Trong trường hợp này, chúng tôi sẽ gọi hàm dựng sẵn wx.MessageBox. Hàm này được gọi với hai tham số. Đầu tiên là đoạn văn bản chúng tôi muốn gửi trong hộp thông báo và hai là tiêu đề cho hộp thông báo. Làm dấu chú thích ở dòng event.Skip() và đặt vào dòng sau.

```
wx.MessageBox('You Clicked
the button', 'Info')
```

Lưu lại và nhấn nút Run (mũi tên màu vàng). Bạn sẽ nhìn thấy một thứ như thế này:



Và khi nhấn nút bạn sẽ thấy như thế này:

Hãy hiểu rằng đây chỉ là cách đơn giản nhất để gọi hộp thông báo. Ngoài ra, bạn có thể thêm nhiều tham số hơn.

Dưới đây là một bản tóm gọn làm thế nào để thay đổi biểu tượng trạng thái trên hộp thông báo (chi tiết hơn vào số tới).

```
#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()
```

wx.ICON_QUESTION
- Hiển thị biểu tượng câu hỏi.

wx.ICON_EXCLAMATION
- Hiển thị một biểu tượng cảnh báo.

wx.ICON_ERROR
- Hiển thị một biểu tượng báo lỗi.

wx.ICON_INFORMATION
- Hiển thị một biểu tượng thông tin

Cách viết:

```
wx.MessageBox('You Clicked
the button', 'Info',
wx.ICON_INFORMATION)
```

Hoặc bất cứ biểu tượng gì bạn muốn dùng cho phù hợp với tình huống. Hơn nữa cũng có nhiều kiểu bố cục các nút mà chúng ta sẽ nói lần tới.

Vì vậy, trong khi chờ cho tới lần tới, hãy thử với các bộ điều khiển, cách bố trí khác nhau và nhiều hơn thế. Chúc vui!

Greg Walters là chủ của RainyDay Solutions, LLC, một công ty tư vấn ở Aurora, Colorado, là lập trình viên từ năm 1972. Ông thích nấu nướng, đi bộ đường dài, nghe nhạc và dành thời gian cho gia đình.

LÀM THẾ NÀO Lập trình Python – Phần 6

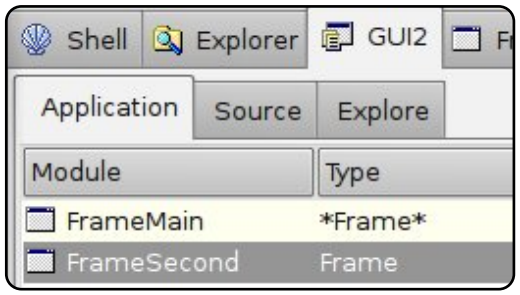
Viết bởi Greg Walters

Tôi hy vọng bạn đã có những trải nghiệm thú vị với Boa Constructor kể từ lần gặp đầu tiên. Chúng ta sẽ bắt đầu một chương trình đơn giản với một cửa sổ và cho phép chúng ta nhấn vào một nút bấm để làm xuất hiện một cửa sổ khác. Lần trước là một bảng thông báo. Lần này, chúng ta sẽ tạo một cửa sổ hoàn toàn độc lập. Điều này có thể rất hữu ích khi thiết kế một ứng dụng có chứa nhiều cửa sổ hoặc khung. Giờ thì chúng ta đi thôi...

Khởi động Boa Constructor và đóng tất cả các tab trong khung soạn thảo trừ tab Shell và Explorer bằng tổ hợp phím Ctrl + W. Điều này bảo đảm chúng ta bắt đầu hoàn toàn từ con số 0. Bây giờ, hãy tạo một dự án mới bằng cách nhấn vào nút wx.App (tham khảo bài viết trước).

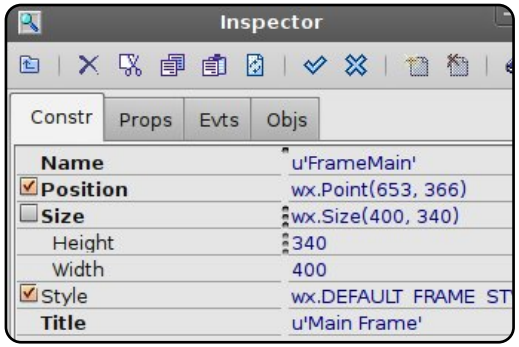
Trước khi làm bất kỳ điều gì, hãy lưu Frame1 với tên "FrameMain.py" và App1 với tên "GUI2.py". Đây là điều quan trọng. Sau khi chọn thẻ GUI2, chuyển qua khung công cụ và thêm một khung khác vào dự án bằng cách nhấn nút wx.Frame (nằm kế nút wx.App). Vẫn trong thẻ GUI2, đảm bảo rằng thẻ Application hiển thị đồng thời cả hai khung dưới cột Module. Giờ trở lại khung mới tạo và lưu lại với tên "FrameSecond.py".

Kế tiếp, mở FrameMain trong chế độ thiết kế. Thêm wx.Panel vào khung. Chỉnh kích thước sao cho nó chiếm toàn bộ khung. Sau đó, chúng ta sẽ thay đổi một vài thuộc tính, điều mà chúng đã không làm ở kỳ trước. Trong khung kiểm tra, chọn Frame1 trong thẻ Objs rồi trong thẻ Constr, sửa tên Title thành "Main Frame" và tên Name thành "FrameMain". Chúng ta sẽ bàn một chút về quy ước đặt tên sau. Hãy thiết lập kích thước 400x340 bằng cách nhấp vào hộp kiểm Size. Nó sẽ thả xuống hai tùy chọn chiều cao và chiều rộng. Chiều cao (Height) là 340 và chiều rộng (Width) là 400.



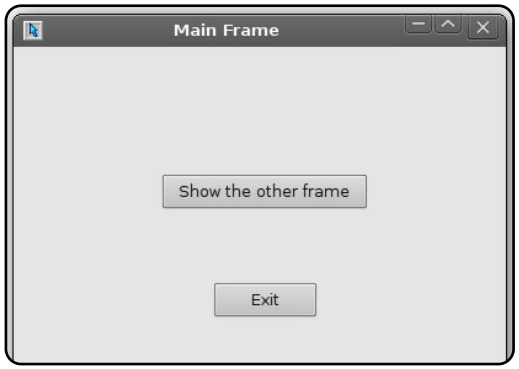
Bây giờ hãy nhấn vào thẻ Props. Nhấn vào thuộc tính Centered và chọn wx.BOTH. Nhấn vào nút duyệt và lưu lại. Bây giờ chạy ứng dụng của bạn bằng cách nhấn vào nút mũi tên màu vàng. Ứng dụng của chúng ta sẽ hiện ra ngay trung tâm màn hình với tiêu đề "Main Frame". Giờ hãy đóng nó lại bằng cách nhấn nút "X" ở bên góc phải của ứng dụng.

Mở khung FrameMain trong chế độ thiết kế. Thêm hai nút wx.Buttons vào khung, cái ở



trên cái ở dưới và gần giữa khung. Chọn nút ở trên, đặt tên là "btnShowNew" và đặt nhãn là "Show the other frame" trong thẻ Constr của khung kiểm tra. Dùng tổ hợp phím Shift + mũi tên để chỉnh kích thước nút sao cho thấy hết nhãn nút và dùng Ctrl + mũi tên để dịch chuyển nút trở lại giữa khung. Chọn nút ở dưới, đặt tên là "btnExit" và nhãn là "Exit". Duyệt, lưu rồi chạy để xem sự thay đổi. Thoát ứng dụng rồi trở lại khâu thiết kế. Chúng ta sẽ thêm sự kiện nhấn nút. Chọn nút phía trên, vào khung kiểm tra, chọn thẻ Evts. Bấm vào ButtonEvent, rồi nhấp đôi vào wx.EVT_BUTTON. Để ý rằng bạn sẽ có dòng "OnBtnShowNewButton" bên dưới. Kế tiếp, chọn nút btnExit. Làm điều tương tự, đảm bảo nó xuất hiện dòng "OnBtnExitButton". Duyệt và lưu. Chuyển qua khung soạn thảo và cuộn xuống dưới cùng.

Hãy chắc rằng bạn có hai phương thức sự kiện vừa được tạo. Dưới đây là hình minh họa cho khung của chúng ta:



Bây giờ là lúc chúng ta thiết kế một khung cửa sổ khác. Mở FrameSecond trong thiết kế. Đặt tên Name là "FrameSecond" và tên Title là "Second Frame". Đặt canh giữa là wx.BOTH. Thêm một wx.Button và chuyển nó vào giữa nhưng gần sát đáy khung. Đặt tên là "btnFSExit", và nhãn là "Exit". Thiết lập sự kiện cho nó. Tiếp theo, thêm một wx.StaticText vào phần nửa trên của khung. Đặt tên là "stHiThere", nhãn là "Hi there... I'm the second form!", thiết lập phông chữ là Sans, PointSize là 14 và Weight là wx.BOLD. Sau đó canh giữa dòng chữ trở lại. Bạn có thể làm điều này bằng cách bỏ chọn thuộc tính Position và chỉnh tọa độ X để dịch qua trái phải, tọa độ Y để dịch lên xuống cho tới khi vừa ý. Duyệt và lưu lại.

Chúng ta vừa kết thúc phần thiết kế, chúng ta sẽ tạo "chất keo" để kết nối tất cả lại với nhau.

Trong khung soạn thảo, nhấn vào thẻ GUI2, sau đó, nhấn vào thẻ Source bên dưới. Dưới dòng ghi là "import FrameMain", hãy thêm "import FrameSecond". Lưu lại thay đổi. Kế đến, chọn thẻ "FrameMain". Bên dưới dòng ghi "import wx", hãy thêm một dòng "import FrameSecond". Tiếp theo, cuộn xuống và tìm dòng ghi "def__init__(self, parent):". Thêm một dòng phía sau dòng "self._init_ctrls(parent)" với nội dung là "self.Fs = FrameSecond.FrameSecond(self)". Bây giờ dưới "def OnBtnShowNewButton(self, event):", đánh dấu chú thích lên dòng "event.Skip()" và thêm hai dòng dưới đây:

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

Cuối cùng, dưới phương thức "OnBtnExitButton", chú thích dòng "event.Skip()", và thêm dòng "self.Close()"

Thế chúng có nghĩa là gì? Thế này, điều đầu tiên chúng ta làm là đảm bảo rằng chương trình nhận biết có hai khung cửa sổ trong nó. Đó là tại sao chúng ta nhập cả hai FrameMain và FrameSecond trong tập tin GUI2. Kế đến chúng ta nhập một tham chiếu đến FrameSecond vào FrameMain để có thể sử dụng sau này. Chúng ta đã khởi tạo nó trong phương thức "_init_". Và trong sự kiện "OnBtnShowNewButton" chúng ta đã bảo nó rằng khi bút được bấm, chúng ta muốn thấy khung thứ hai được hiện ra và khung thứ nhất biến mất. Cuối cùng chúng ta có một câu lệnh để đóng chương trình khi nút Exit được nhấn.

Bây giờ, hãy chuyển sang viết lệnh cho FrameSecond. Chỉ một vài thay đổi nhỏ thôi. Dưới phương thức "_init_", thêm dòng "self.parent = parent" để tạo biến self.parent. Cuối cùng bên dưới sự kiện nhấn nút FSExitButton, chú thích dòng "event.Skip()" và thêm hai dòng:

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

Còn nhớ chúng ta đã giấu khung đầu tiên khi khung thứ hai xuất hiện không? Giờ chúng ta phải làm ngược lại. Chúng ta giấu khung thứ hai đi. Cuối cùng, lưu thay đổi của bạn.

Đây là toàn bộ mã lệnh để bạn kiểm tra lại mọi thứ (trang này và trang kế tiếp):

Bây giờ bạn có thể chạy chương trình. Nếu mọi thứ diễn ra tốt đẹp, bạn có thể nhấn vào nút btnShowNew và thấy khung đầu tiên biến mất ngay khi khung thứ hai hiện ra. Nhấn nút Exit trên khung thứ hai sẽ làm khung này biến mất và khung ban đầu sẽ xuất hiện trở lại. Nhấn nút Exit trên khung chính này sẽ kết thúc chương trình.

Tôi đã hứa với bạn là chúng ta sẽ bàn về quy ước đặt tên. Hãy nhớ lại, có phải chúng ta đã bàn về việc chú thích mã

lệnh? Vậy thì, bằng cách dùng những tên gọi hợp lý cho những bộ điều khiển của GUI, mã lệnh của bạn hầu như được tự động giải thích. Nếu bạn chỉ để tên các bộ điều khiển như là staticText1 hay button1 hay gì đi nữa, khi bạn tạo một khung phức tạp với rất nhiều thành phần điều khiển, đặc biệt nếu có rất nhiều ô nhập liệu hoặc nút bấm, thì việc đặt tên có ý nghĩa rất quan trọng. Nó có thể không mấy quan trọng nếu chỉ có bạn là người xem mã nguồn, nhưng để người khác có thể tiếp tục phát triển công việc của bạn, những cái tên đúng đắn sẽ giúp họ rất nhiều. Vì thế, hãy sử dụng cách đặt tên tương tự như sau:

GUI2 code:

```
#!/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()
```

FrameMain code:

```
#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)
```

FrameMain Code (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()
```

FrameSecond code:

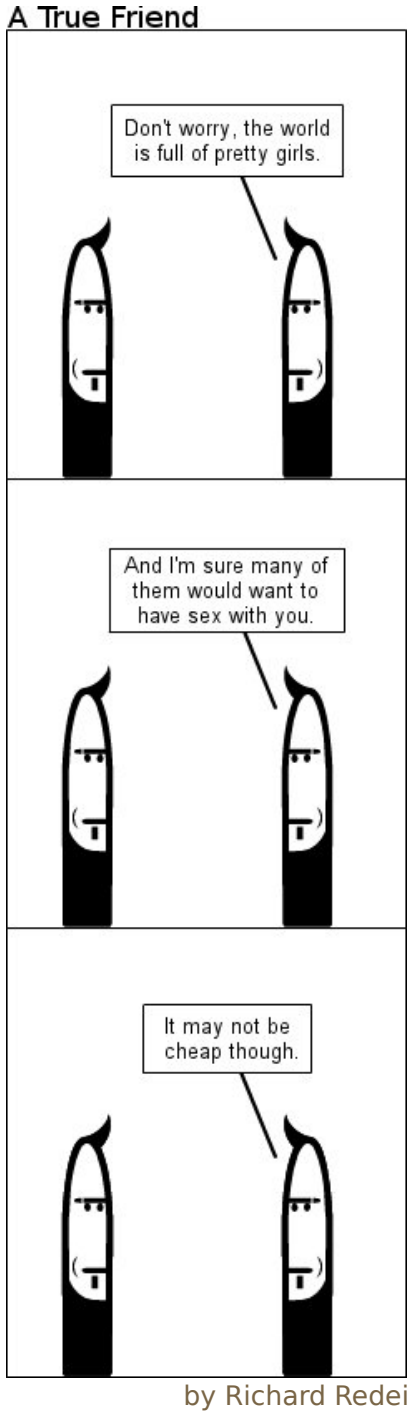
```
#Boa:Frame:FrameSecond

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()
```

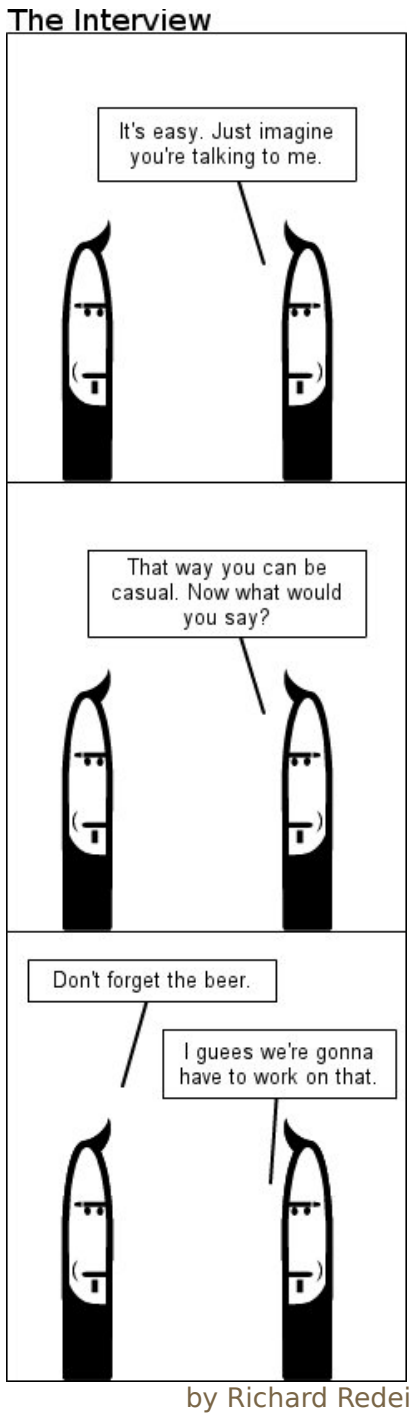
```
Control type - Name prefix
Static text - st_
Button - btn_
Text Box - txt_
Check Box - chk_
Radio Button - rb_
Frame - Frm_ or Frame_
```

Bạn có thể có những ý tưởng riêng về quy ước đặt tên trong quá trình trở thành lập trình viên và trong một vài trường hợp, sếp hay người chủ của bạn có lẽ đã lập ra sẵn những quy ước.

Lần tới, chúng ta sẽ tạm biệt lập trình giao diện đồ họa để tập trung vào lập trình cơ sở dữ liệu. Trong lúc này, hãy cài đặt python-apsw và python-mysqldb lên máy của bạn. Bạn cũng sẽ cần sqlite và sqlitebrowser cho SQLite. Nếu bạn cũng muốn trải nghiệm với MySql, đó là một ý tưởng không tồi. Tất cả đều có sẵn trong Synaptic.

Greg Walters là chủ của RainyDay Solutions, LLC, một công ty tư vấn ở Aurora, Colorado, là lập trình viên từ năm 1972. Ông thích nấu nướng, đi bộ đường dài, nghe nhạc và dành thời gian cho gia đình.





LÀM THẾ NÀO Lập trình Python – Phần 7

Viết bởi Greg Walters

Xin chào các chàng trai và cô gái. Đã đến giờ kể chuyện. Các bạn đã tìm một chỗ ngồi thật thoải mái chưa? Sẵn sàng rồi chứ? Thế thì chúng ta bắt đầu!

Ngày xửa ngày xưa, thế giới tràn ngập giấy. Giấy, giấy ở khắp nơi. Người ta phải xây những ngôi nhà đặc biệt cho chúng. Họ gọi đó là tủ hồ sơ, những chiếc hộp kim loại to lớn đó chiếm đầy hết căn phòng này đến căn phòng nọ, từ cơ quan cho đến gia đình chỉ để chứa giấy. Trong mỗi tủ hồ sơ có những thứ gọi là túi hồ sơ dùng để nhóm các giấy tờ có liên quan lại với nhau. Nhưng sau một thời gian, chúng cũng chật cứng hoặc mục nát vì để quá lâu hoặc được tra cứu quá nhiều.

Để sử dụng đúng cách những tủ hồ sơ đó, người ta cần phải có trình độ bậc phổ thông. Phải mất cả ngày để lục tìm tất cả những giấy tờ được xếp trong những tủ hồ sơ khác nhau. Công việc bị trì trệ rất nhiều. Đó quả là thời khắc cực kỳ đen tối trong lịch sử loài người.

Rồi một ngày, trên đỉnh của một ngọn núi nào đó (tôi nghĩ đó là Colorado, tôi không chắc lắm), xuất hiện một cô tiên xinh đẹp. Cô khoác trên người một màu xanh lấp lánh ánh bạc, với đôi cánh lộng lẫy và mái tóc trắng muốt, và cô chỉ cao khoảng 30 cm. Tên cô ta, dù tin hay không, là See-Quill (đọc như SQL). Một cái tên buồn cười phải không? Sao cũng được, See-Quill nói rằng cô có thể giải quyết tất cả mọi vấn đề từ giấy tờ, tủ hồ sơ và thời gian, với điều kiện mọi người phải tin tưởng vào cô ta và máy điện toán. Cô gọi phép màu nhiệm của mình là "Cơ sở dữ liệu". Cô nói rằng "Cơ sở dữ liệu" có thể thay thế toàn bộ hệ thống hồ sơ hiện tại. Một số người đã làm theo và cuộc sống của họ trở nên rất hạnh phúc. Một số khác thì không, và cuộc sống của họ vẫn không thay đổi, lạc lõng giữa hàng núi giấy.

Tuy nhiên, tất cả những lời hứa hẹn của cô tiên đều kèm theo điều kiện. Lần này, điều kiện là những ai muốn sử dụng quyền năng của See-Quill đều phải học một ngôn ngữ mới. Nó không quá khó để học. Thực tế, ngôn ngữ này giống với cái mọi người thường sử dụng. Chỉ có một chút khác biệt trong cách nói, và cần phải suy nghĩ thật kỹ trước khi nói để sử dụng quyền năng của See-Quill.

Một ngày nọ, một chàng thanh niên tò mò tên là User đến gặp See-Quill. Anh ta rất cảm kích trước vẻ đẹp của cô và nói "See-Quill ơi, làm ơn chỉ cho tôi cách sử dụng quyền năng của cô đi." See-Quill đáp lại rằng cô sẽ chỉ cho anh ta.

Cô nói với chàng trai: "Trước tiên, anh phải biết thông tin của anh được sắp xếp như thế nào. Vui lòng cho em xem những giấy tờ của anh."

Vì còn trẻ, User chỉ có một vài tờ giấy. See-Quill nói với anh ta: "User này, hiện tại anh có thể sống với những xấp giấy và những tập hồ sơ. Tuy nhiên, em có thể tiên đoán rằng một ngày kia, anh sẽ có nhiều giấy đến nỗi nếu xếp chồng chúng lên nhau thì chúng sẽ cao gấp 15 lần anh đó. Nhưng anh đừng lo vì đã có quyền năng của em mà."

Thế là User và See-Quill làm việc cùng nhau và họ đã tạo nên một thứ gọi là "cơ sở dữ liệu" (một thuật ngữ kỹ thuật của thần tiên) và User sống rất hạnh phúc cho đến cuối đời.

Hết truyện.

Dĩ nhiên, truyện kể không hoàn toàn đúng thực tế. Tuy nhiên, việc sử dụng cơ sở dữ liệu và SQL có thể làm cho cuộc sống chúng ta dễ dàng hơn. Trong bài này chúng ta sẽ tìm hiểu một vài truy vấn SQL đơn giản và làm thế nào để áp dụng vào chương trình. Một vài người có thể nghĩ rằng đây có thể không phải là một cách "đúng" hay là cách "tốt nhất" nhưng đó là một cách hợp lý. Vậy thì hãy

bắt đầu.

Cơ sở dữ liệu giống như cái tủ hồ sơ trong câu chuyện vừa rồi. Bảng dữ liệu giống như tập hồ sơ. Một bản ghi độc lập trong bảng giống như một tờ giấy. Mỗi mẫu thông tin được gọi là trường. Tất cả được sắp xếp rất gọn gàng phải không? Bạn dùng câu lệnh SQL để thao tác với dữ liệu. SQL nghĩa là ngôn ngữ truy vấn có cấu trúc (Structured Query Language) và được thiết kế để đơn giản hóa cách sử dụng cơ sở dữ liệu. Nhưng trong thực tế, nó có thể trở nên rất phức tạp. Chúng ta sẽ giữ mức độ đơn giản trong giai đoạn này.

Khi bắt đầu xây dựng bất cứ dự án nào, chúng ta cần tạo một sơ đồ. Hãy lấy ví dụ về phiếu công thức nấu ăn, đó là một ví dụ hay bởi vì chúng ta sẽ tạo nên một cơ sở dữ liệu về công thức. Ở nhà tôi, công thức nằm ở nhiều dạng: tấm cạc 3x5 in, tờ giấy 8x10 in, khăn ăn với công thức in trên đó, trang tạp chí và kể cả những dạng lạ lùng khác. Chúng có thể được tìm thấy trên sách, hộp, tủ, v.v.. Tuy nhiên, chúng đều có một điểm chung: định dạng của chúng. Trong hầu hết trường hợp, chúng ta thấy phần đầu là tên món ăn và có thể là số khẩu phần ăn và xuất xứ món ăn. Phần thân chứa danh sách nguyên liệu và phần cuối chứa cách nấu, trình tự nấu, thời gian nấu, v.v.. Chúng ta sẽ sử dụng định dạng chung này như một kiểu mẫu cho cơ sở dữ liệu của chúng ta. Chúng ta sẽ chia dự án này thành hai phần: hôm nay chúng ta sẽ tạo cơ sở dữ liệu và lần tới sẽ tạo ứng dụng để có thể tra cứu và cập nhật dữ liệu.

Lấy một ví dụ. Giả sử chúng ta có một công thức món ăn ở phía bên phải.

Hãy chú ý trình tự mà chúng ta vừa bàn. Khi chúng ta thiết kế cơ sở dữ liệu, chúng ta có thể dự kiến chứa tất cả thông tin của công thức chỉ trong một bản ghi. Tuy nhiên, nó trở nên nặng nề và khó khăn trong quản lý sau này. Thay vì vậy, chúng ta sẽ sử dụng phiếu nấu ăn như một kiểu mẫu. Một bảng sẽ chứa phần đầu, tức là thông tin tổng quát của món ăn, bảng khác sẽ lưu phần thân, nguyên liệu, và một bảng khác nữa để chứa phần cuối, cách làm món ăn.

Đảm bảo bạn đã cài đặt SQLite và APSW. SQLite là một bộ máy quản lý cơ sở dữ liệu nhỏ hoạt động không cần phải có một máy chủ cơ sở dữ liệu riêng, rất phù hợp với dự án nhỏ của chúng ta. Mọi thứ bạn học được ở đây có thể áp dụng cho hệ thống cơ sở dữ liệu lớn hơn như MySQL, ... Một điểm hay nữa ở SQLite là nó sử dụng giới hạn các kiểu dữ liệu. Đó là kiểu Text, Numeric, Blob và Integer Primary Key. Như các bạn đã biết, kiểu Text cho phép chứa bất kỳ thông tin văn bản nào. Những nguyên liệu, chỉ dẫn, tựa món ăn đều ở dạng Text, kể cả đôi khi chúng chứa những con số. Kiểu Numeric cho phép chứa những con số, có thể là số nguyên, số thực hay dấu chấm động. Blob là kiểu dữ liệu nhị phân, ví dụ như hình ảnh. Integer Primary Key hơi đặc biệt một tí. SQLite dùng nó để ghi tự động một giá trị số

Spanish Rice

Serves: 4

Source: Greg Walters

Ingredients:
1 cup parboiled Rice (uncooked)
1 pound Hamburger
2 cups Water
1 8 oz can Tomato Sauce
1 small Onion chopped
1 clove Garlic chopped
1 tablespoon Ground Cumin
1 teaspoon Ground Oregano
Salt and Pepper to taste
Salsa to taste

Instructions:
Brown hamburger.

Add all other ingredients.

Bring to boil.

Stir, lower to simmer and cover.

Cook for 20 minutes.

Do not look, do not touch.

Stir and serve.

nguyên duy nhất. Điều này sẽ quan trọng trong chốc lát nữa.

APSW là từ viết tắt của chữ Another Python SQLite Wrapper, nó cho phép kết nối dễ dàng với SQLite. Bây giờ hãy xem xét một vài cách tạo truy vấn SQL.

Để lấy thông tin từ cơ sở dữ liệu, bạn sẽ dùng câu lệnh SELECT. Cú pháp là:

```
SELECT [what] FROM [which table(s)] WHERE [Constraints]
```

Vậy, nếu chúng ta muốn lấy toàn bộ các trường trong bảng Recipes (công thức), chúng ta sẽ viết:

```
SELECT * FROM Recipes
```

Nếu bạn muốn lấy dữ liệu theo khóa chính của nó, bạn phải biết giá trị của khóa chính đó (pkID, trong ví dụ này) và thêm lệnh WHERE vào câu lệnh, như thế này:

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

Cũng đơn giản, phải không? Cứ như là ngôn ngữ hàng ngày ấy. Bây giờ, hãy giả định rằng chúng ta chỉ muốn lấy tên của món ăn và số khẩu phần ăn của tất cả các công thức. Dễ ợt. Tất cả những gì chúng ta làm là thêm một danh sách các trường vào trong câu truy vấn SELECT:

```
SELECT name, servings FROM Recipes
```

Để thêm thông tin, chúng ta dùng câu lệnh INSERT INTO. Cú pháp là:

```
INSERT INTO [table name] (field list) VALUES (values to insert)
```

Vậy, để thêm một công thức vào bảng Recipes, câu lệnh sẽ là:

```
INSERT INTO Recipes (name, serves, source) VALUES ("Tacos", 4, "Greg"")
```

Để xóa một bản ghi, chúng ta có thể dùng

```
DELETE FROM Recipes WHERE pkID = 10
```

Ngoài ra cũng có câu lệnh cập nhật UPDATE, nhưng chúng ta sẽ nói về nó vào một dịp khác.

Nói thêm về SELECT

Trong trường hợp cơ sở dữ liệu của chúng ta, chúng ta có 3 bảng liên kết với nhau nhờ các recipeID trỏ về pkID trong bảng Recipes. Giả sử chúng ta muốn lấy thông tin cách nấu của một món ăn. Chúng ta có thể làm như sau:

```
SELECT Recipes.name,
Recipes.serves,
Recipes.source,
Instructions.instructions
FROM Recipes LEFT JOIN
Instructions ON (Recipes.pkID
= Instructions.recipeID)
WHERE Recipes.pkID = 1
```

Tuy nhiên, câu lệnh trên quá dài và dư thừa. Chúng ta có thể dùng những từ thay thế (alias) như sau:

```
SELECT r.name, r.servings,
r.source, i.instructions
FROM Recipes r LEFT JOIN
Instructions i ON (r.pkID =
i.recipeID)
WHERE r.pkID = 1
```

Nó ngắn hơn mà vẫn dễ đọc. Bây giờ chúng ta sẽ viết một chương trình nhỏ để tạo cơ sở dữ liệu, các bảng và nhập một vài dữ liệu đơn giản vào bảng để có thể làm việc. Trong tương lai, chúng ta có thể tích hợp nó vào trong chương trình đầy đủ, nhưng trong ví dụ này, chúng ta sẽ tách thành chương trình riêng lẻ. Đây là chương trình chỉ chạy một lần, nếu bạn chạy lần thứ hai, bạn sẽ gặp lỗi khi tạo bảng. Ngoài ra, chúng ta có thể đặt nó vào giữa cặp lệnh "try...catch", nhưng chúng ta sẽ làm điều đó vào một lúc khác.

Chúng ta hãy bắt đầu bằng việc nhập mô-đun APSW.

```
import apsw
```

Điều tiếp theo chúng ta cần làm là tạo kết nối tới cơ sở dữ liệu. Nó sẽ được đặt ở cùng thư mục với ứng dụng của ta. Khi chúng ta tạo kết nối, SQLite sẽ tự động dò tìm xem cơ sở dữ liệu đã tồn tại hay chưa. Nếu chưa, nó sẽ tạo cho chúng ta. Một khi chúng ta có kết nối, chúng ta cần cái gọi là con trỏ. Cái đó sẽ tạo một cơ chế để chúng ta thao tác với cơ sở dữ liệu. Vì vậy nên nhớ rằng chúng ta cần một kết nối và một con trỏ. Chúng được tạo ra như sau:

```
# Opening/creating database
connection=apsw.Connection("c
ookbook.db3")
cursor=connection.cursor()
```

Vậy là chúng ta đã có một kết nối và một con trỏ. Bây giờ chúng ta cần tạo ra bảng. Sẽ có 3 bảng trong ứng dụng. Một để chứa thông tin chung của món ăn, một chứa toàn bộ hướng dẫn nấu nướng và một chứa thành phần nguyên liệu. Chúng ta không thể làm điều đó chỉ trong một bảng phải không? Có chứ, dĩ nhiên là được, nhưng bạn thấy đó, nó sẽ làm cho bảng rất phức tạp và chứa nhiều thông tin trùng lắp.

Chúng ta có thể xem xét cấu trúc bảng như thế này. Mỗi cột là một bảng:

```
RECIPES
------------
pkID (Integer Primary Key)
name (Text)
source (Text)
serves (Text)
```

```
INSTRUCTIONS
----------------------
pkID(Integer Primary Key)
recipeID (Integer)
instructions (Text)
```

```
INGREDIENTS
--------------------
pkID (Integer Primary Key)
recipeID (Integer)
ingredients (Text)
```

Mỗi bảng có một trường gọi là pkID. Đó là khóa chính và là giá trị có tính duy nhất trong bảng. Điều này là quan trọng vì dữ liệu trong bảng không bao giờ được phép có 2 dòng giống hệt nhau. Nó có kiểu số nguyên và được cấp phát tự động bởi bộ máy cơ sở dữ liệu. Bạn có thể làm mà không cần đến nó không? Có thể, nhưng bạn dễ nhầm lẫn dẫn đến trùng lặp số id. Trong trường hợp của bảng Recipes, chúng ta sẽ dùng con số này để biết cách nấu nào và nguyên liệu nào sẽ kết hợp với món ăn đó.

Đầu tiên, chúng ta sẽ đặt thông tin tên, nguồn gốc và khẩu phần ăn vào bảng Recipes. pkID được cấp phát tự động. Nếu như chúng ta đặt dòng thông tin đầu tiên vào bảng, hệ thống sẽ cấp phát cho pkID giá trị là 1. Chúng ta sẽ sử dụng giá trị này để liên kết các thông tin liên quan ở các bảng khác. Bảng Instructions cũng đơn giản. Nó chỉ chứa một văn bản dài các hướng dẫn, chỉ số pkID của riêng nó và một chỉ số trỏ về một công thức trong bảng Recipes. Bảng Ingredients phức tạp hơn một tí ở chỗ chúng ta có mỗi bản ghi cho mỗi nguyên liệu cũng như chỉ số pkID của riêng nó và chỉ số trỏ về món ăn trong bảng Recipes.

Vậy để tạo bảng Recipes, chúng ta định nghĩa một biến kiểu chuỗi tên là sql và gán câu lệnh tạo bảng cho nó:

```
sql = 'CREATE TABLE Recipes
(pkID INTEGER PRIMARY KEY,
name TEXT, serves TEXT,
source TEXT)'
```

Kế đến chúng ta yêu cầu APSW thực hiện câu lệnh:

```
cursor.execute(sql)
```

Bây giờ chúng ta tạo những bảng khác:

```
sql = 'CREATE TABLE
Instructions (pkID INTEGER
PRIMARY KEY, instructions
TEXT, recipeID NUMERIC)'

cursor.execute(sql)

sql = 'CREATE TABLE
Ingredients (pkID INTEGER
PRIMARY KEY, ingredients
TEXT, recipeID NUMERIC)'

cursor.execute(sql)
```

Một khi tất cả các bảng đã được tạo, chúng ta sử dụng câu lệnh INSERT INTO để nhập từng khối dữ liệu vào mỗi bảng.

Nhớ rằng pkID được cấp phát tự động, vì thế chúng ta không tính đến nó trong danh sách các trường trong câu lệnh insert. Và vì chúng ta sẽ xác định tên của trường nên chúng có thể được đặt theo bất kỳ trật tự nào, không bắt buộc phải theo trình tự đã tạo ra trong bảng. Miễn là chúng ta nhập đúng tên của trường, mọi thứ sẽ diễn ra suông sẻ. Câu lệnh insert vào bảng Recipes như sau:

```
sql = 'INSERT INTO Recipes
(name, serves, source) VALUES
("Spanish Rice", 4, "Greg
Walters")'

cursor.execute(sql)
```

Kế đến chúng ta cần biết giá trị đã được gán vào pkID trong

bảng Recipes. Chúng ta có thể làm điều đó bằng câu lệnh đơn giản sau:

```
SELECT last_insert_rowid()
```

Tuy nhiên, không thể sử dụng trực tiếp dễ dàng như vậy. Cần phải dùng một loạt các câu lệnh sau:

```
sql = "SELECT
last_insert_rowid()"

cursor.execute(sql)

for x in cursor.execute(sql):
    lastid = x[0]
```

Tại sao lại như vậy? Chà, khi chúng ta lấy dữ liệu từ APSW, nó trả về dưới dạng tuple. Đây là điều chúng ta chưa từng đề cập đến. Giải thích ngắn gọn thì tuple giống như một danh sách (list), nhưng không thể thay đổi được. Nhiều người hiếm dùng tuple, nhưng một số thì dùng thường xuyên; dùng hay không là tùy bạn. Dòng lệnh cuối cùng có nghĩa là chúng ta muốn lấy giá trị trả về đầu tiên. Chúng ta dùng vòng lặp 'for' để lấy giá trị trong biến tuple x. Hiểu chứ? OK, chúng ta tiếp tục...

Bước tiếp theo chúng ta sẽ tạo câu lệnh insert cho bảng Instructions:

```
sql = 'INSERT INTO
Instructions (recipeID,
instructions) VALUES ( %s,
"Brown hamburger. Stir in all
other ingredients. Bring to a
boil. Stir. Lower to simmer.
Cover and cook for 20 minutes
or until all liquid is
absorbed.")' % lastid

cursor.execute(sql)
```

Lưu ý rằng chúng ta đang dùng sự thay thế biến (%s) để đưa pkID của món ăn (lastid) vào trong câu lệnh sql. Cuối cùng, chúng ta cần chèn từng nguyên liệu vào trong bảng Ingredients. Tôi sẽ cho bạn xem một ví dụ:

```
sql = 'INSERT INTO
Ingredients (recipeID,
ingredients) VALUES ( %s, "1
cup parboiled Rice
(uncooked)")' % lastid

cursor.execute(sql)
```

Cuối cùng, lưu tất cả lại và chạy trong một cửa sổ dòng lệnh để tạo cơ sở dữ liệu. Cũng không quá khó đúng không? Lần tới sẽ phức tạp hơn một chút đấy.

Nếu bạn muốn chép toàn bộ mã nguồn, tôi đã để nó trên trang web của tôi, bạn có thể tải về tại đây: www.thedesignatedgeek.com

Kỳ tới, chúng ta sẽ áp dụng những điều đã học trong suốt quá trình để tạo một giao diện với các trình đơn truy cập vào các công thức. Nó cho phép xem toàn bộ các công thức dưới dạng danh sách, xem một công thức bất kỳ, tìm kiếm và thêm bớt công thức.

Tôi khuyên bạn nên dành chút thời gian để đọc tài liệu về lập trình SQL. Bạn sẽ thấy vui vì điều đó.

Greg Walters là chủ của RainyDay Solutions, LLC, một công ty tư vấn ở Aurora, Colorado, là lập trình viên từ năm 1972. Ông thích nấu nướng, đi bộ đường dài, nghe nhạc và dành thời gian cho gia đình.

LÀM THẾ NÀO
Viết bởi Greg Walters

Lập trình Python – Phần 8

Chúng ta sẽ tiếp tục lập trình cơ sở dữ liệu (CSDL) về công thức nấu ăn mà chúng ta đã làm ở phần 7. Đây sẽ là một bài dài, với nhiều dòng lệnh vì vậy hãy "giữ vững tay chèo" và đừng vội nản chí nhé. Chúng ta đã tạo CSDL. Bây giờ chúng ta muốn hiển thị phần nội dung, thêm vào hay bớt đi. Vậy chúng ta làm điều đó như thế nào? Chúng ta sẽ bắt đầu với một ứng dụng chạy trong cửa sổ dòng lệnh, vì thế chúng ta cần tạo ra một trình đơn (menu). Chúng ta cũng sẽ tạo một lớp chứa các hàm thực thi CSDL. Hãy bắt đầu bằng một đoạn mã như ở cột bên phải.

Bây giờ chúng ta sẽ bố cục trình đơn. Trình đơn sẽ là một vòng lặp hiển thị danh sách các lựa chọn mà người dùng có thể yêu cầu. Chúng ta dùng dòng lặp while. Viết hàm Menu giống đoạn mã ở cột bên phải phía dưới.

Kế đến, chúng ta hoàn thành trình đơn bằng một cấu trúc if|elif|else, xem ví dụ mẫu ở đầu trang kế.

Hãy nhìn lại hàm tạo trình đơn của chúng ta. Chúng ta bắt đầu bằng việc in những mệnh lệnh mà người dùng có thể yêu cầu. Chúng ta đặt biến loop là True và dùng hàm while để tiếp tục lặp cho đến khi loop bằng False. Chúng ta dùng lệnh raw_input() để chờ người dùng đưa ra một lựa chọn và khối cấu trúc if để xử lý yêu cầu người dùng. Trước khi chúng ta chạy thử chương trình, chúng ta cần tạo một hàm __init__ trong lớp.

```
def __init__(seft):
    pass
```

Giờ thì lưu chương trình lại tại nơi mà bạn đã tạo CSDL và chạy nó. Bạn sẽ thấy điều tương tự như khung bên phải trang sau.

Nó chỉ đơn giản hiện lên trình đơn hết lần này đến lần khác, cho đến khi bạn nhập "0"

```
#!/usr/bin/python
#-------------------------------------------------------
# Cookbook.py
# Created for Beginning Programming Using Python #8
# and Full Circle Magazine
#-------------------------------------------------------
import apsw
import string
import webbrowser

class Cookbook():
    def __init__(self):
        pass

def Menu():
    cbk = Cookbook() # Initialize the class

Menu()
```

```
def Menu():
    cbk = Cookbook() # Initialize the class
    loop = True
    while loop == True:
        print
'=================================================='
        print '                RECIPE DATABASE'
        print
'=================================================='
        print ' 1 - Show All Recipes'
        print ' 2 - Search for a recipe'
        print ' 3 - Show a Recipe'
        print ' 4 - Delete a recipe'
        print ' 5 - Add a recipe'
        print ' 6 - Print a recipe'
        print ' 0 - Exit'
        print
'=================================================='
        response = raw_input('Enter a selection -> ')
```

```
    if response == '1': # Show all recipes
        pass
    elif response == '2': # Search for a recipe
        pass
    elif response == '3': # Show a single recipe
        pass
    elif response == '4': # Delete Recipe
        pass
    elif response == '5': # Add a recipe
        pass
    elif response == '6': # Print a recipe
        pass
    elif response == '0': # Exit the program
        print 'Goodbye'
        loop = False
    else:
        print 'Unrecognized command.  Try again.'
```

```
/usr/bin/python -u
"/home/greg/python_examples/APSW/cookbook/cookbook_stub.py"
==================================================
                RECIPE DATABASE
==================================================
1 - Show All Recipes
2 - Search for a recipe
3 - Show a Recipe
4 - Delete a recipe
5 - Add a recipe
6 - Print a recipe
0 - Exit
==================================================
Enter a selection ->
```

thì nó sẽ in câu "Goodbye" và thoát. Đến đây, chúng ta có thể bắt đầu viết các hàm trong lớp Cookbook. Chúng ta sẽ cần một hàm để hiển thị tất cả thông tin có trong bảng Recipes, một hàm cho phép bạn tìm kiếm công thức, một hàm để xóa công thức, một hàm để thêm công thức và một hàm để in công thức từ một máy in mặc định. Hàm PrintAllRecipes không cần bất cứ tham số nào khác ngoại trừ tham số (self), tương tự cho hàm SearchforRecipe và hàm EnterNew. Hàm PrintSingleRecipe, DeleteRecipe và PrintOut đều cần biết công thức nào để xử lý, vậy chúng cần một tham số, ta gọi nó là "which". Hãy dùng lệnh pass để kết thúc mỗi hàm. Bên trong lớp Cookbook, tạo ra các hàm:

```
def PrintAllRecipes(self):
    pass
def SearchForRecipe(self):
    pass
def
PrintSingleRecipe(self,which)
:
    pass
def DeleteRecipe(self,which):
    pass
def EnterNew(self):
    pass
def PrintOut(self,which):
    pass
```

Đối với một số mục trong trình đơn, chúng ta muốn in ra danh sách tất cả các công thức trong bảng Recipes để người dùng có thể dò tìm trong danh sách. Những mục chọn đó có thể là 1, 3, 4 và 6. Vậy, hãy sửa đổi hàm menu các mục đó, thay lệnh pass bằng cbk.PrintAllRecipes(). Các bạn có thể tham khảo đoạn mã chỉnh sửa ở trang sau.

Một điều cần làm nữa là thiết lập hàm __init__. Thay thế đoạn mã với những dòng sau:

```
def __init__(self):
  global connection
  global cursor
  self.totalcount = 0

connection=apsw.Connection("c
ookbook.db3")
  cursor=connection.cursor()
```

Đầu tiên, chúng ta tạo hai biến toàn cục cho kết nối và con trỏ. Chúng ta có thể truy xuất chúng ở bất kỳ đâu trong lớp cookbook. Kế đến, chúng ta tạo một biến tên self.totalcount để đếm số lượng công thức. Chúng ta sẽ sử dụng nó sau. Cuối cùng chúng ta tạo kết nối và con trỏ.

Bước tiếp theo là hoàn thiện hàm PrintAllRecipes() trong lớp Cookbook. Bởi vì chúng ta có hai biến toàn cục cho kết nối và con trỏ nên chúng ta không cần phải tạo lại chúng trong mỗi hàm. Tiếp, chúng ta muốn hiển thị lên màn hình những tiêu đề công thức. Chúng ta sẽ dùng ký tự thay thế "%s" và lệnh canh lề trái để tạo khoảng cách. Chúng ta muốn nó giống như thế này:

```
Item   Name   Serves   Source
-------------------------
```

```
        if response == '1': # Show all recipes
            cbk.PrintAllRecipes()
        elif response == '2': # Search for a recipe
            pass
        elif response == '3': # Show a single recipe
            cbk.PrintAllRecipes()
        elif response == '4': # Delete Recipe
            cbk.PrintAllRecipes()
        elif response == '5': # Add a recipe
            pass
        elif response == '6': # Print a recipe
            cbk.PrintAllRecipes()
        elif response == '0': # Exit the program
            print 'Goodbye'
            loop = False
        else:
            print 'Unrecognized command.  Try again.'
```

Cuối cùng, chúng ta cần tạo câu lệnh SQL, truy vấn CSDL và hiển thị kết quả.

Hầu hết đã được đề cập ở bài viết trước.

```
sql = 'SELECT * FROM Recipes'
cntr = 0
for x in cursor.execute(sql):
    cntr += 1
    print '%s %s %s %s'
%(str(x[0]).rjust(5),x[1].lju
st(30),x[2].ljust(20),x[3].lj
ust(30))
    print '-------------'
    self.totalcount = cntr
```

Biến cntr sẽ đếm số lượng công thức hiển thị cho người dùng. Hàm của chúng ta đã xong. Khung bên phải là đoạn mã đầy đủ của hàm để bạn tham khảo khi cần.

Lưu ý rằng chúng ta đang sử dụng kết quả kiểu tuple được trả về từ lệnh cursor.execute của APSW. Chúng ta hiển thị số pkID của mỗi công thức ở cột Item. Điều này giúp chúng ta chọn đúng công thức sau này. Khi bạn chạy chương trình, bạn sẽ thấy trình đơn, khi bạn chọn mục số 1, bạn sẽ nhận được kết quả như hình ở đầu trang sau.

Đó là điều chúng ta muốn, ngoại trừ nếu bạn đang chạy ứng dụng trong Dr.Python hoặc tương tự vậy, chương trình sẽ không có những khoảng dừng. Hãy thêm một khoảng dừng cho đến khi người dùng nhấn một nút để họ có thể theo dõi trong một vài giây. Trong lúc đó, hãy hiển thị tổng số công thức từ biến mà chúng ta đã tạo ra trước đó. Hãy thêm vào dưới lựa chọn 1 của trình đơn:

```
print 'Total Recipes - %s'
%cbk.totalcount
print '----------------------
----------------------------
-'
res = raw_input('Press A Key
-> ')
```

Chúng ta sẽ bỏ qua mục số 2 (Search for a recipe) một lát và xử lý mục số 3 (Show a single recipe). Hãy lưu tâm đến phần trình đơn trước. Chúng ta sẽ hiển thị danh sách các công thức, như mục số 1, và hỏi người dùng chọn một. Để đảm bảo không mắc lỗi vì người dùng nhập sai, chúng ta sẽ in câu hỏi cho người dùng (Select a recipe =>), nếu người dùng nhập đúng, chúng ta sẽ gọi hàm PrintSingleRecipe() trong lớp Cookbook với mã pkID từ bảng Recipes. Nếu đầu vào không phải là con số, nó sẽ báo lỗi ValueError, chúng ta sẽ bắt nó bằng lệnh except ValueError: như khung bên phải trang kế.

Kế tiếp, chúng ta sẽ làm việc với hàm PrintSingleRecipe trong lớp Cookbook. Vì đã có kết nối và con trỏ rồi nên chúng ta tạo câu truy vấn SQL. Trong trường hợp này, chúng ta dùng `sql = 'SELECT * FROM Recipes WHERE pkID = %s' %str(which)`

```
def PrintAllRecipes(self):
        print '%s %s %s %s' %('Item'.ljust(5),
'Name'.ljust(30), 'Serves'.ljust(20),
'Source'.ljust(30))
        print '---------------------------------'
        sql = 'SELECT * FROM Recipes'
        cntr = 0
        for x in cursor.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.totalcount = cntr
```

```
Enter a selection -> 1
Item  Name                              Serves              Source
-------------------------------------------------------------------------------
    1 Spanish Rice                      4                   Greg
    2 Pickled Pepper-Onion Relish       9 half pints        Complete Guide to Home Canning
-------------------------------------------------------------------------------
==================================================
               RECIPE DATABASE
==================================================
1 - Show All Recipes
2 - Search for a recipe
3 - Show a Recipe
4 - Delete a recipe
5 - Add a recipe
6 - Print a recipe
0 - Exit
==================================================
Enter a selection ->
```

với which là giá trị chúng ta muốn tìm. Sau đó chúng ta xuất ra màn hình (một cách thân thiện) kết quả tuple trả về từ APSW. Với trường hợp này, chúng ta dùng x như là một biến tuple tổng hợp, có các thành phần với chỉ số index (nằm giữa cặp dấu ngoặc vuông). Bởi vì bố cục của bảng có dạng pkID/name/serves/source, chúng ta có thể dùng x[0],x[1],x[2] và x[3] để biểu diễn thành phần. Sau đó, chúng ta muốn chọn hết mọi nguyên liệu từ bảng Ingredients nơi mà recipeID (khóa ngoại liên kết đến bảng Recipes) bằng với pkID vừa mới sử dụng. Chúng ta chạy vòng lặp trong kết quả tuple trả về, in từng nguyên liệu, tiếp theo chúng ta lấy cách nấu trong bảng Instructions như cách đã làm với bảng Ingredients. Cuối cùng, chúng ta đợi người dùng nhấn một nút để họ có thể thấy công thức trên màn hình. Đoạn mã mẫu nằm ở trang kế.

Giờ đây chúng ta đã hoàn tất hai trên sáu hàm. Hãy xử lý hàm tìm kiếm, cũng bắt đầu từ trình đơn. Thật may, lần này chúng ta chỉ gọi hàm tìm kiếm trong lớp, vậy hãy thay thế lệnh pass bằng:

```
cbk.SearchForRecipe()
```

Phần còn lại là đi viết hàm tìm kiếm. Trong lớp Cookbook, thay thế nội dung trong hàm SearchForRecipe bằng đoạn mã

```
try:
    res = int(raw_input('Select a Recipe -> '))
    if res <= cbk.totalcount:
        cbk.PrintSingleRecipe(res)
    elif res == cbk.totalcount + 1:
        print 'Back To Menu...'
    else:
        print 'Unrecognized command.  Returning to menu.'
except ValueError:
    print 'Not a number...back to menu.'
```

nằm ở trang 39.

Có rất nhiều thứ ở đây. Sau khi đã tạo kết nối và con trỏ, chúng ta hiển thị trình đơn tìm kiếm. Chúng ta sẽ cho người dùng 3 cách để tìm kiếm và một lối thoát trở về trình đơn chính. Chúng ta có thể để người dùng tìm một từ trong tên công thức, một từ trong nguồn gốc công thức hoặc một từ trong danh sách nguyên liệu. Do vậy, chúng ta không thể chỉ dùng hàm hiển thị mới vừa tạo mà phải tạo một hàm hiển thị đặc thù. Hai mục đầu tiên sử dụng câu lệnh SELECT đơn giản với một chút bổ sung. Chúng ta sẽ sử dụng câu lệnh định tính "like". Nếu chúng ta quen sử dụng một bộ duyệt truy vấn như SQLite Database Browser, câu lệnh like dùng một cặp ký tự "%". Vậy để tìm một tên công thức có chứa từ "rice", câu

truy vấn sẽ là:

```
SELECT * FROM Recipes WHERE
name like '%rice%'
```

Tuy nhiên, vì ký tự "%" cũng là ký tự thay thế chuỗi, chúng ta sẽ dùng %%. Tồi tệ hơn nữa, chúng ta sẽ dùng ký tự thay thế để chèn chuỗi người dùng tìm kiếm. Vì thế, chúng ta phải ghi là '%%%s%%'. Xin lỗi nếu nó làm bạn rối mù lên. Ở câu truy vấn thứ 3 còn có câu lệnh Join. Hãy quan sát kỹ hơn:

```
sql = "SELECT
r.pkid,r.name,r.serves,r.sour
ce,i.ingredients FROM Recipes
r Left Join ingredients i on
(r.pkid = i.recipeid) WHERE
i.ingredients like '%%%s%%'
GROUP BY r.pkid" %response
```

Chúng ta chọn mọi thứ trong bảng Recipes và chọn nguyên liệu trong bảng Ingredients, kết nối giữa bảng nguyên liệu và bảng công thức dựa trên mã số recipeID bằng với pkID, rồi tìm kiếm nguyên liệu dựa trên câu lệnh like và cuối cùng là nhóm kết quả theo mã pkID để tránh trùng lắp. Nếu bạn còn nhớ, chúng ta có hai thứ tiêu trong công thức thứ hai (nước sốt hành và tiêu), một xanh và một đỏ. Điều đó có thể gây lẫn lộn cho chúng ta. Trong trình đơn, chúng ta có dùng:

```
searchin = raw_input('Enter
Search Type -> ')
if searchin != '4':
```

có nghĩa là: nếu searchin (giá trị người dùng nhập vào) KHÔNG bằng 4 thì thực thi các lựa chọn, nếu bằng 4 thì không làm gì cả. Lưu ý rằng tôi dùng "!=" thay vì "<>" để biểu diễn sự không bằng nhau. Mặc dù Python 2.x chấp nhận cả hai, nhưng trong Python 3.x, nó sẽ báo lỗi sai cú pháp. Chúng ta sẽ bàn về những thay đổi trong Python 3.x ở một bài viết khác. Bây giờ hãy dùng "!=" để giúp bạn nhanh chóng bắt kịp với Python 3.x trong tương lai. Sau cùng, chúng ta xuất thông tin ra màn hình theo cách dễ đọc cho người dùng. Hãy xem người dùng thấy gì ở cột bên phải trang 40.

Bạn có thể thấy nội dung chương trình xuất ra đẹp đẽ thế nào. Bây giờ người dùng có thể quay trở lại trình đơn chính và dùng mục thứ 3 để truy xuất bất cứ công thức nào họ muốn xem. Kế tiếp chúng ta sẽ thêm

```
  def PrintSingleRecipe(self,which):
        sql = 'SELECT * FROM Recipes WHERE pkID = %s' %
str(which)
        print
'~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~'
        for x in cursor.execute(sql):
            recipeid = x[0]
            print "Title: " + x[1]
            print "Serves: " + x[2]
            print "Source: " + x[3]
        print
'~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~'
        sql = 'SELECT * FROM Ingredients WHERE RecipeID =
%s' % recipeid
        print 'Ingredient List:'
        for x in cursor.execute(sql):
            print x[1]
        print ''
        print 'Instructions:'
        sql = 'SELECT * FROM Instructions WHERE RecipeID
= %s' % recipeid
        for x in cursor.execute(sql):
            print x[1]
        print
'~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~'
        resp = raw_input('Press A Key -> ')
```

công thức vào CSDL. Chúng ta cũng thêm một dòng vào trong hàm menu để gọi tới hàm EnterNew:

```
cbk.EnterNew()
```

Mã nguồn để thay thế cho hàm EnterNew() trong lớp Cookbook có thể tìm thấy ở trang 43.

Chúng ta bắt đầu bằng việc định nghĩa một danh sách tên "ings", có nghĩa là nguyên liệu. Sau đó chúng ta yêu cầu người dùng nhập vào tiêu đề, nguồn gốc và số khẩu phần. Kế đến chúng ta mở một vòng lặp, nhập từng nguyên liệu, chèn vào danh sách ings. Nếu người dùng nhập 0, thì sẽ thoát vòng lặp và tiếp tục nhập hướng dẫn cách làm. Sau đó chúng ta hiển thị nội dung công thức và yêu

```
    def SearchForRecipe(self):
        # print the search menu
        print '------------------------------'
        print ' Search in'
        print '------------------------------'
        print ' 1 - Recipe Name'
        print ' 2 - Recipe Source'
        print ' 3 - Ingredients'
        print ' 4 - Exit'
        searchin = raw_input('Enter Search Type -> ')
        if searchin != '4':
            if searchin == '1':
                search = 'Recipe Name'
            elif searchin == '2':
                search = 'Recipe Source'
            elif searchin == '3':
                search = 'Ingredients'
            parm = searchin
            response = raw_input('Search for what in %s (blank to exit) -> ' % search)
            if parm == '1': # Recipe Name
                sql = "SELECT * FROM Recipes WHERE name like '%%%s%%'" %response
            elif parm == '2': # Recipe Source
                sql = "SELECT * FROM Recipes WHERE source like '%%%s%%'" %response
            elif parm == '3': # Ingredients
                sql = "SELECT r.pkid,r.name,r.serves,r.source,i.ingredients FROM Recipes r Left Join Ingredients i on
(r.pkid = i.recipeid) WHERE i.ingredients like '%%%s%%' GROUP BY r.pkid" %response
            try:
                if parm == '3':
                    print '%s %s %s %s %s'
%('Item'.ljust(5),'Name'.ljust(30),'Serves'.ljust(20),'Source'.ljust(30),'Ingredient'.ljust(30))
                    print '-------------------------------------------------------------------------------'
                else:
                    print '%s %s %s %s' %('Item'.ljust(5),'Name'.ljust(30),'Serves'.ljust(20),'Source'.ljust(30))
                    print '-------------------------------------------------------------------------------'
                for x in cursor.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 'An Error Occured'
            print '-------------------------------------------------------------------------------'
            inkey = raw_input('Press a key')
```

```
Enter a selection -> 2
------------------------------
Search in
------------------------------
1 - Recipe Name
2 - Recipe Source
3 - Ingredients
4 - Exit
Enter Search Type -> 1
Search for what in Recipe Name (blank to exit) -> rice
Item  Name                          Serves               Source
-------------------------------------------------------------------------------
    1 Spanish Rice                  4                    Greg
-------------------------------------------------------------------------------
Press a key
```

Easy enough. Now for the ingredient search...

```
Enter a selection -> 2
------------------------------
Search in
------------------------------
1 - Recipe Name
2 - Recipe Source
3 - Ingredients
4 - Exit
Enter Search Type -> 3
Search for what in Ingredients (blank to exit) -> onion
Item  Name                  Serves           Source                          Ingredient
-------------------------------------------------------------------------------
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
-------------------------------------------------------------------------------
Press a key
```

cầu người dùng duyệt qua trước khi lưu lại. Chúng ta dùng câu lệnh INSERT INTO như lần trước và trở lại trình đơn. Một điều chúng ta phải cẩn thận là dấu nháy đơn trong dữ liệu nhập vào. Bình thường thì nó không thành vấn đề trong danh sách nguyên liệu hay cách nấu, nhưng lại có thể xảy ra đối với trường tiêu đề hay nguồn gốc. Chúng ta cần thêm một ký tự thoát cho bất kỳ dấu nháy đơn. Chúng ta làm điều đó bằng hàm string.replace, đó là lý do tại sao chúng ta nhập thư viện string.

Trong hàm menu, đặt đoạn mã nằm ở bên phải phía trên vào dưới mục số 4.

Rồi trong lớp Cookbook, dùng đoạn mã ở bên phải phía dưới cho hàm DeleteRecipe().

Chúng ta sẽ nhanh chóng tìm hiểu hàm xóa. Đầu tiên chúng ta yêu cầu người dùng nhập vào mã công thức muốn xóa (trong hàm menu), và truyền mã pkID vào hàm xóa. Kế đến, chúng ta yêu cầu người dùng có chắc muốn xóa công thức đó không. Nếu câu trả lời là "Y" (string.upper(resp) == 'Y'), thì chúng ta tạo câu lệnh sql xóa. Lưu ý rằng lúc này chúng ta phải xóa các bản ghi ở cả ba bảng. Chúng ta có thể chỉ xóa bản ghi ở bảng Recipes, nhưng dữ liệu liên kết ở 2 bảng còn lại sẽ trở nên lạc lõng và điều này là không tốt. Khi chúng ta xóa bản ghi ở bảng Recipes, chúng ta sử dụng trường pkID. Trong 2 bảng kia, chúng ta dùng trường recipeID.

Cuối cùng, chúng ta xử lý hàm in công thức. Chúng ta sẽ tạo một tập tin HTML thật đơn giản, mở nó bằng trình duyệt mặc định và in nó từ đó. Đó là tại sao chúng ta nhập thư viện webbrowser. Trong hàm menu, ở mục số 6, chèn đoạn mã ở đầu trang sau.

Một lần nữa, chúng ta hiển thị danh sách của các công thức ra giấy, và cho phép người dùng chọn một cái mà họ muốn in. Chúng ta gọi hàm PrintOut trong lớp Cookbook. Đó là đoạn mã thứ hai ở trang sau.

Chúng ta bắt đầu bằng lệnh fi = open([filename],'w') để tạo ra tập tin. Sau đó chúng ta lấy thông tin từ bảng Recipes và ghi xuống tập tin với lệnh fi.write. Chúng ta dùng cặp thẻ <H1></H1> cho tiêu đề, cặp <H2></H2> cho số khẩu phần và nguồn gốc. Kế đến chúng ta dùng cặp cho danh sách nguyên liệu, và ghi cách làm. Phần còn lại là những câu truy vấn đơn giản chúng ta đã học. Sau cùng, chúng ta đóng tập tin lại bằng lệnh fi.close() và dùng webbrowser([filename]) để mở tập tin vừa mới tạo. Người dùng có thể in thông tin ra giấy từ

```
        cbk.PrintAllRecipes()
        print '0 - Return To Menu'
        try:
            res = int(raw_input('Select a Recipe to DELETE
 or 0 to exit -> '))
            if res != 0:
                cbk.DeleteRecipe(res)
            elif res == 0:
                print 'Back To Menu...'
            else:
                print 'Unrecognized command.  Returning to
 menu.'
        except ValueError:
             print 'Not a number...back to menu.'
```

```
    def DeleteRecipe(self,which):
        resp = raw_input('Are You SURE you want to Delete
this record? (Y/n) -> ')
        if string.upper(resp) == 'Y':
            sql = "DELETE FROM Recipes WHERE pkID = %s" %
str(which)
            cursor.execute(sql)
            sql = "DELETE FROM Instructions WHERE
recipeID = %s" % str(which)
            cursor.execute(sql)
            sql = "DELETE FROM Ingredients WHERE recipeID
= %s" % str(which)
            cursor.execute(sql)
            print "Recipe information DELETED"
            resp = raw_input('Press A Key -> ')
        else:
            print "Delete Aborted - Returning to menu"
```

trình duyệt nếu họ muốn.

Quoa! Đó là một chương trình "vĩ đại" nhất của chúng ta ngày hôm nay. Nếu bạn không muốn gõ lại hết tất cả hay gặp vấn đề gì đó thì bạn có thể tải mã nguồn hoàn chỉnh tại đây:

```
        cbk.PrintAllRecipes()
        print '0 - Return To Menu'
        try:
            res = int(raw_input('Select a Recipe to DELETE or 0 to exit -> '))
            if res != 0:
                cbk.PrintOut(res)
            elif res == 0:
                print 'Back To Menu...'
            else:
                print 'Unrecognized command.  Returning to menu.'
        except ValueError:
            print 'Not a number...back to menu.'
```

```
    def PrintOut(self,which):
        fi = open('recipeprint.html','w')
        sql = "SELECT * FROM Recipes WHERE pkID = %s" % which
        for x in cursor.execute(sql):
            RecipeName = x[1]
            RecipeSource = x[3]
            RecipeServings = x[2]
        fi.write("<H1>%s</H1>" % RecipeName)
        fi.write("<H2>Source: %s</H2>" % RecipeSource)
        fi.write("<H2>Servings: %s</H2>" % RecipeServings)
        fi.write("<H3> Ingredient List: </H3>")
        sql = 'SELECT * FROM Ingredients WHERE RecipeID = %s' % which
        for x in cursor.execute(sql):
            fi.write("<li>%s</li>" % x[1])
        fi.write("<H3>Instructions:</H3>")
        sql = 'SELECT * FROM Instructions WHERE RecipeID = %s' % which
        for x in cursor.execute(sql):
            fi.write(x[1])
        fi.close()
        webbrowser.open('recipeprint.html')
        print "Done"
```

Greg Walters là chủ của RainyDay Solutions, LLC, một công ty tư vấn ở Aurora, Colorado, là lập trình viên từ năm 1972. Ông thích nấu nướng, đi bộ đường dài, nghe nhạc và dành thời gian cho gia đình.

```
def EnterNew(self):

    ings = []
    recipename = ''
    recipesource = ''
    recipeserves = ''
    instructions = ''
    lastid = 0

    resp = raw_input('Enter Recipe Title (Blank line to exit) -> ')

    if resp != '' :  # continue

        if string.find(resp,"'"):
            recipename = resp.replace("'","\'")
        else:
            recipename = resp

        print "RecipeName will be %s" % recipename

        resp = raw_input('Enter Recipe Source -> ')

        if string.find(resp,"'"):
            recipesource = resp.replace("'","\'")
        else:
            recipesource = resp

        resp = raw_input('Enter number of servings -> ')

        if string.find(resp,"'"):
            recipeserves = resp.replace("'","\'")
        else:
            recipeserves = resp

        print 'Now we will enter the ingredient list.'

        cont = True
        while cont == True:
            ing = raw_input('Enter Ingredient ("0" to exit) -> ')
            if ing != '0':
                ings.append(ing)
            else:
                cont = False
```

```
            resp = raw_input('Enter Instructions -> ')
            instructions = resp

            print '~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~'
            print "Here's what we have so far"
            print "Title: %s" % recipename
            print "Source: %s" % recipesource
            print "Serves: %s" % recipeserves
            print "Ingredients:"
            for x in ings:
                print x
            print "Instructions: %s" % instructions
            print '~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~'

            resp = raw_input("OK to save? (Y/n) -> ")

            if string.upper(resp) != 'N':
                # Write the Recipe Record
                sql = 'INSERT INTO Recipes (name,serves,source) VALUES ("%s","%s","%s")' %(recipename, recipeserves,
recipesource)
                cursor.execute(sql)

                # Get the new Recipe pkID
                sql = "SELECT last_insert_rowid()"
                cursor.execute(sql)
                for x in cursor.execute(sql):
                    lastid = x[0]
                    print "last id = %s" % lastid

                # Write the Ingredients Record
                for x in ings:
                    sql = 'INSERT INTO Ingredients (recipeID,ingredients) VALUES (%s,"%s")' % (lastid,x)
                    cursor.execute(sql)

                # Write the Instructions records
                sql = 'INSERT INTO Instructions (recipeID,instructions) VALUES( %s,"%s")' %(lastid,instructions)
                cursor.execute(sql)

                # Prompt the user that we are done
                print 'Done'
            else:
                print 'Save aborted'
```