

Tổng quan về Python

Trong chương này chúng ta sẽ tìm hiểu sơ qua về ngôn ngữ lập trình Python, về lịch sử và một số đặc điểm của nó.

Python là gì?

Python là một ngôn ngữ lập trình bậc cao, thông dịch, hướng đối tượng, đa mục đích và cũng là một ngôn ngữ lập trình động.

Cú pháp của Python là khá dễ dàng để học và ngôn ngữ này cũng mạnh mẽ và linh hoạt không kém các ngôn ngữ khác trong việc phát triển các ứng dụng. Python hỗ trợ mẫu đa lập trình, bao gồm lập trình hướng đối tượng, lập trình hàm và mệnh lệnh hoặc là các phong cách lập trình theo thủ tục.

Python không chỉ làm việc trên lĩnh vực đặc biệt như lập trình web, và đó là tại sao ngôn ngữ này là đa mục đích bởi vì nó có thể được sử dụng với web, enterprise, 3D CAD, ...

Bạn không cần sử dụng các kiểu dữ liệu để khai báo biến bởi vì kiểu của nó là động, vì thế bạn có thể viết `a=15` để khai báo một giá trị nguyên trong một biến.

Với Python, việc phát triển ứng dụng và debug trở nên nhanh hơn bởi vì không cần đến bước biên dịch và chu trình edit-test-debug của Python là rất nhanh.

Các đặc điểm của Python

Dưới đây là một số đặc điểm chính của Python:

- **Dễ dàng để sử dụng:** Python là một ngôn ngữ bậc cao rất dễ dàng để sử dụng. Python có một số lượng từ khóa ít hơn, cấu trúc của Python đơn giản hơn và cú pháp của Python được định nghĩa khá rõ ràng, ... Tất cả các điều này là Python thực sự trở thành một ngôn ngữ thân thiện với lập trình viên.
- Bạn có thể **đọc code của Python khá dễ dàng.** Phần code của Python được định nghĩa khá rõ ràng và rành mạch.
- Python có một **thư viện chuẩn khá rộng lớn.** Thư viện này dễ dàng tương thích và tích hợp với UNIX, Windows, và Macintosh.

- Python là một **ngôn ngữ thông dịch**. Trình thông dịch thực thi code theo từng dòng (và bạn không cần phải biên dịch ra file chạy), điều này giúp cho quá trình debug trở nên dễ dàng hơn và đây cũng là yếu tố khá quan trọng giúp Python thu hút được nhiều người học và trở nên khá phổ biến.
- Python cũng là một ngôn ngữ lập trình **hướng đối tượng**. Ngoài ra, Python còn hỗ trợ các phương thức lập trình theo hàm và theo cấu trúc.

Ngoài các đặc điểm trên, Python còn khá nhiều đặc điểm khác như hỗ trợ lập trình GUI, mã nguồn mở, có thể tích hợp với các ngôn ngữ lập trình khác, ...

Lịch sử của Python

Python được phát triển bởi Guido Van Rossum vào cuối những năm 80 và đầu những năm 90 tại Viện toán-tin ở Hà Lan. Python kế thừa từ nhiều ngôn ngữ như ABC, Module-3, C, C++, Unix Shell, ...

Ngôn ngữ Python được cập nhật khá thường xuyên để thêm các tính năng và hỗ trợ mới. Phiên bản mới nhất hiện nay của Python là **Python 3.3** được công bố vào 29/9/2012 với nguyên tắc chủ đạo là "bỏ cách làm việc cũ nhằm hạn chế trùng lặp về mặc chức năng của Python".

Hướng dẫn cài đặt Python

Trước khi cài đặt Python, bạn cần tải Python từ: <https://www.python.org/downloads/>.

Sau khi đã tải Python về máy, thì việc cài đặt Python cũng giống như cài đặt các phần mềm khác như bộ Visual Studio của Microsoft, bạn cứ nhấn next, next và next.

Nếu bạn chỉ tải và cài cái này thì có thể biên dịch các python file bằng terminal. Để sử dụng đầy đủ các tính năng tạo project, tạo file, biên dịch, ... bạn nên cài thêm IDE pycharm. Bạn theo link sau để tải: <https://www.jetbrains.com/pycharm/download/>.

Dưới đây là cách thiết lập path cho một số hệ điều hành phổ biến:

Thiết lập path trên Windows

Để thêm thư mục Python tới path cho một phiên cụ thể trong Windows thì tại dòng nhắc lệnh, bạn gõ path %path%;C:\Python và nhấn Enter. Bạn cũng cần chú ý một số biến môi trường sau:

PYTHONPATH: Nó có vai trò như PATH. Biến này nói cho Trình thông dịch của Python nơi để đặt các file được nhập vào một chương trình. Nó cũng bao thư mục thư viện nguồn và các thư mục chứa source code của Python.

PYTHONSTARTUP: Nó gồm path của một file khởi tạo chứa source code của Python và được thực thi mỗi khi bạn bắt đầu trình thông dịch. Trong Unix, nó có tên là .pythonrc.py.

PYTHONCASEOK: được sử dụng trong Windows để chỉ dẫn cho Python để tìm các kết nối không phân biệt kiểu chữ trong một lệnh quan trọng.

PYTHONHOME: Nó là một path tìm kiếm thay thế, và thường được nhúng trong các thư mục PYTHONSTARTUP hoặc PYTHONPATH.

Thiết lập path trên Unix/Linux

Trong **csh shell**: gõ setenv PATH "\$PATH:/usr/local/bin/python" và nhấn Enter.

Trong **bash shell**: gõ export PATH="\$PATH:/usr/local/bin/python" và nhấn Enter.

Trong sh hoặc **ksh shell**: bạn gõ PATH="\$PATH:/usr/local/bin/python" và nhấn Enter.

Bạn nên nhớ là /usr/local/bin/python là path của thư mục Python.

Chương trình Hello World trong Python

Chương này sẽ trình bày cách viết chương trình Python đầu tiên để in ra dòng chữ "Hello World" cũng như cách thực thi chương trình Python trong các chế độ khác nhau.

Chương trình Python để in "Hello World"

Dưới đây là đoạn code đơn giản để in ra dòng chữ "Hello World" trong Python.

```
>>> a="Hello World"
>>> print a
Hello World
>>>
```

Giải thích:

- Ở đây, chúng ta đang sử dụng IDE để viết Python code. Phần giải thích chi tiết để chạy chương trình sẽ được trình bày ngay dưới đây.
- Một biến được định nghĩa với tên là a và giữ "Hello World".
- Lệnh print được sử dụng in ra nội dung. Nếu bạn đang sử dụng phiên bản Python mới nhất (phiên bản 3.3 chẳng hạn) thì lệnh print đã được thay đổi thành hàm print(), tức là trong phiên bản mới nhất bạn cần phải thêm các dấu ngoặc đơn vào như dưới đây chẳng hạn:

```
>>> a=("Hello World")
>>> print a
Hello World
>>>
```

Cách thực thi Python trong chế độ tương tác

Bạn triệu hồi dòng nhắc lệnh như sau:

```
$ python
Python 2.4.3 (#1, Nov 11 2015, 13:34:43)
[GCC 4.1.2 20080704 (Red Hat 4.1.2-48)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Sau đó, bạn gõ dòng lệnh sau và nhấn Enter:

```
>>> print "Hello World"
```

Nếu bạn đang chạy trên phiên bản Python mới nhất, bạn cần sử dụng hàm print với dấu ngoặc đơn, như **print ("Hello World");**.

Cách thực thi Python trong chế độ script

Sử dụng chế độ script, bạn cần viết Python code trong một file riêng rẽ bởi sử dụng **bất cứ** trình soạn thảo nào trong hệ điều hành của bạn. Sau đó, bạn lưu nó với đuôi **.py** và mở dòng nhắc lệnh để thực thi.

Giả sử bạn gõ source code sau trong một test.py file:

```
print "Hello World"
```

Nếu bạn đã có trình thông dịch của Python được thiết lập trong biến PATH, bây giờ bạn thử chạy chương trình trên như sau:

```
$ python test.py
```

Lệnh trên sẽ cho kết quả:

```
Hello World
```

Bây giờ, chúng ta thử một cách khác để thực thi một Python script. Sau đây là test.py file đã được sửa đổi:

```
print "Hello World"
```

Giả sử bạn có trình thông dịch của Python có sẵn trong thư mục /usr/bin, thì bạn có thể chạy chương trình trên như sau:

```
$ chmod +x test.py      # Dong nay giup file co the thuc thi  
$ ./test.py
```

Lệnh trên sẽ cho kết quả:

Hello World

Cú pháp Python cơ bản

Chương này sẽ trình bày khái quát cho bạn về cú pháp Python cơ bản. Mục đích của chương là giúp bạn làm quen dần các khái niệm và thuật ngữ được sử dụng trong Python từ đó bạn có thể rút ra điểm giống và khác nhau với một số ngôn ngữ lập trình khác.

Định danh (identifier) trong Python

Một định danh (identifier) trong Python là một tên được sử dụng để nhận diện một biến, một hàm, một lớp, hoặc một đối tượng. Một định danh bắt đầu với một chữ cái từ A tới Z hoặc từ a tới z hoặc một dấu gạch dưới (_) được sau bởi 0 hoặc nhiều ký tự, dấu gạch dưới hoặc các ký số (từ 0 tới 9).

Python không hỗ trợ các Punctuation char chẵng hạn như @, \$ và % bên trong các định danh. Python là một ngôn ngữ lập trình phân biệt kiểu chữ, do đó **Vietjack** và **vietjack** là hai định danh khác nhau trong Python. Dưới đây là một số qui tắc nên được sử dụng trong khi đặt tên các định danh:

- Một định danh là một dãy ký tự hoặc ký số.
- Không có ký tự đặc biệt nào được sử dụng (ngoại trừ dấu gạch dưới) như một định danh. Ký tự đầu tiên có thể là chữ cái, dấu gạch dưới, nhưng không được sử dụng ký số làm ký tự đầu tiên.
- Từ khóa không nên được sử dụng như là một tên định danh (phần dưới sẽ trình bày về khác từ khóa này).
- Tên lớp bắt đầu với một chữ cái hoa. Tất cả định danh khác bắt đầu với một chữ cái thường.
- Bắt đầu một định danh với một dấu gạch dưới đơn chỉ rằng định danh đó là private.
- Bắt đầu một định danh với hai dấu gạch dưới chỉ rằng định danh đó thực sự là private.
- Nếu định danh cũng kết thúc với hai dấu gạch dưới, thì định danh này là một tên đặc biệt được định nghĩa bởi ngôn ngữ (ví dụ như `__init__` chẵng hạn).

Các từ khóa trong Python

Bảng dưới liệt kê các từ khóa trong Python. Đây là các từ dành riêng và bạn không thể sử dụng chúng như là các hằng, biến hoặc cho bất kỳ tên định danh nào. Tất cả từ khóa trong Python là chỉ ở dạng chữ thường.

and	exec	not
assert	finally	or
break	for	pass
class	from	print
continue	global	raise
def	if	return
del	import	try
elif	in	while
else	is	with
except	lambda	yield

Dòng lệnh và độ thụt dòng lệnh trong Python

Python không cung cấp các dấu ngoặc ôm ({}) để chỉ các khối code cho định nghĩa lớp hoặc hàm hoặc điều khiển luồng. Các khối code được nhận biết bởi độ thụt dòng code (indentation) trong Python và đây là điều bắt buộc.

Số khoảng trắng trong độ thụt dòng là biến đổi, nhưng tất cả các lệnh bên trong khối phải được thụt cùng một số lượng khoảng trắng như nhau. Ví dụ:

```
if True:  
    print "True"  
  
else:  
    print "False"
```

Tuy nhiên, khối sau sẽ tạo ra một lỗi:

```
if True:  
    print "Answer"  
    print "True"  
  
else:  
    print "Answer"  
    print "False"
```

Do đó, trong Python thì tất cả các dòng tiếp nhau mà được thụt đầu dòng với cùng lượng khoảng trắng như nhau sẽ tạo nên một khối. Trong ví dụ tiếp theo sẽ có các khối lệnh đa dạng:

Ghi chú: Bạn không cần cố hiểu vấn đề này ngay lập tức, bạn chỉ cần hiểu các khối code khác nhau ngay cả khi chúng không có các dấu ngoặc ôm. Đây chính là điểm khác nhau giữa Python và ngôn ngữ khác.

```
import sys  
  
try:  
    # open file stream  
    file = open(file_name, "w")  
  
except IOError:  
    print "There was an error writing to", file_name  
    sys.exit()  
  
print "Enter '", file_finish,  
print "' When finished"  
  
while file_text != file_finish:  
    file_text = raw_input("Enter text: ")  
  
    if file_text == file_finish:
```

```
# close the file
file.close
break

file.write(file_text)
file.write("\n")

file.close()

file_name = raw_input("Enter filename: ")

if len(file_name) == 0:
    print "Next time please enter something"
    sys.exit()

try:
    file = open(file_name, "r")
except IOError:
    print "There was an error reading file"
    sys.exit()

file_text = file.read()
file.close()
print file_text
```

Các lệnh trên nhiều dòng trong Python

Các lệnh trong Python có một nét đặc trưng là kết thúc với một newline. Tuy nhiên, Python cho phép sử dụng ký tự \ để chỉ rõ sự liên tục dòng. Ví dụ:

```
total = item_one + \
        item_two + \
        item_three
```

Các lệnh được chứa bên trong các dấu ngoặc [], {}, hoặc () thì không cần sử dụng ký tự \. Ví dụ:

```
days = [ 'Monday', 'Tuesday', 'Wednesday',
         'Thursday', 'Friday']
```

Trích dẫn trong Python

Python chấp nhận trích dẫn đơn ('), kép ("") và trích dẫn tam ("" hoặc "") để biểu thị các hằng chuỗi, miễn là các trích dẫn này có cùng kiểu mở và đóng.

Trích dẫn tam được sử dụng để trải rộng chuỗi được trích dẫn qua nhiều dòng. Dưới đây là tất cả các trích dẫn hợp lệ:

```
word = 'word'

sentence = "This is a sentence."

paragraph = """This is a paragraph. It is
made up of multiple lines and sentences."""
```

Comment trong Python

Python hỗ trợ hai kiểu comment đó là comment đơn dòng và đa dòng. Trong Python, một dấu #, mà không ở bên trong một hằng chuỗi nào, bắt đầu một comment đơn dòng. Tất cả ký tự ở sau dấu # và kéo dài cho đến hết dòng đó thì được coi là một comment và được bỏ qua bởi trình thông dịch. Ví dụ:

```
# First comment
print "Hello, Python!" # second comment
```

Chương trình trên sẽ cho kết quả:

```
Hello, Python!
```

Bạn cũng có thể gõ một comment trên cùng dòng với một lệnh hoặc biểu thức như sau:

```
name = "Madisetti" # This is again comment
```

Bạn có thể comment trên nhiều dòng như sau:

```
# This is a comment.
# This is a comment, too.
# This is a comment, too.
# I said that already.
```

Python cũng hỗ trợ kiểu comment thứ hai, đó là kiểu comment đa dòng được cho bên trong các trích dẫn tam, ví dụ:

```
#single line comment
```

```
print "Hello Python"  
"""This is  
multiline comment"""
```

Sử dụng dòng trống trong Python

Một dòng mà chỉ chứa các khoảng trống trắng whitespace, có thể với một comment, thì được xem như là một dòng trống và Python hoàn toàn bỏ qua nó.

Trong một phiên thông dịch trong chế độ tương tác, bạn phải nhập một dòng trống để kết thúc một lệnh đa dòng.

Các lệnh đa dòng trên một dòng đơn trong Python

Dấu chấm phẩy (;) cho phép xuất hiện nhiều lệnh trên một dòng đơn. Tất cả các lệnh được cung cấp này không bắt đầu một khối code mới. Dưới đây là ví dụ:

```
import sys; x = 'foo'; sys.stdout.write(x + '\n')
```

Các nhóm lệnh đa dòng (còn được gọi là suite) trong Python

Một nhóm các lệnh đơn, mà tạo một khối code đơn, được gọi là **suite** trong Python. Các lệnh phức hợp như if, while, def, và class cần một dòng header và một suite.

Các dòng header bắt đầu lệnh (với từ khóa) và kết thúc với một dấu hai chấm (:) và được theo sau bởi một hoặc nhiều dòng để tạo nên một suite. Ví dụ như:

```
if expression :  
    suite  
elif expression :  
    suite  
else :  
    suite
```

Tham số dòng lệnh trong Python

Nhiều chương trình có thể được chạy để cung cấp cho bạn một số thông tin cơ bản về cách chúng nên được chạy. Python cho bạn khả năng để làm điều này với -h:

```
$ python -h

usage: python [option] ... [-c cmd | -m mod | file | -] [arg] ...

Options and arguments (and corresponding environment variables):

-c cmd : program passed in as string (terminates option list)
-d      : debug output from parser (also PYTHONDEBUG=x)
-E      : ignore environment variables (such as PYTHONPATH)
-h      : print this help message and exit

[ etc. ]
```

Bạn cũng có thể lập trình cho script của mình theo cái cách mà nó nên chấp nhận các tùy chọn khác nhau tùy theo cách bạn thiết lập. Để tìm hiểu thêm về tham số dòng lệnh, bạn có thể tham khảo chương Tham số dòng lệnh trong Python. (Mình đề nghị bạn nên tìm hiểu chương này sau khi bạn đã tìm hiểu qua về các khái niệm còn lại của Python.)

Ngoài ra, một điều cần nói đến đó là khi bạn gặp phải trường hợp chương trình hiển thị dòng nhắc sau:

```
raw_input("\n\nPress the enter key to exit.")
```

Lệnh này nói rằng bạn hãy nhấn phím Enter để thoát. Ở đây, "\n\n" là để tạo hai newline trước khi hiển thị dòng thực sự. Khi người dùng nhấn phím enter, thì chương trình kết thúc. Lệnh này sẽ đợi cho đến khi nào bạn thực hiện một hành động nào đó, và điều này giữ cho cửa sổ console của bạn mở tới khi bạn tiếp tục thực hiện hành động.

Tham số dòng lệnh trong Python

Python cung cấp **getopt** Module giúp bạn phân tích cú pháp các tùy chọn và tham số dòng lệnh.

```
$ python test.py arg1 arg2 arg3
```

sys Module trong Python cung cấp sự truy cập tới bất kỳ tham số dòng lệnh nào thông qua **sys.argv**. Phục vụ hai mục đích:

- **sys.argv** là danh sách các tham số dòng lệnh.
- **len(sys.argv)** là số tham số dòng lệnh.

Ví dụ

Ví dụ

```
import sys

print 'So tham so:', len(sys.argv), 'tham so.'
print 'Danh sach tham so:', str(sys.argv)
```

Bây giờ chạy script trên như sau:

```
$ python test.py arg1 arg2 arg3
```

Kết quả là:

```
So tham so: 4 tham so.
Danh sach tham so: ['test.py', 'arg1', 'arg2', 'arg3']
```

Ghi chú: tham số đầu tiên luôn luôn là tên script và nó cũng được đếm trong số tham số.

Parse các tham số dòng lệnh trong Python

Python cung cấp **getopt** Module giúp bạn phân tích cú pháp các tùy chọn và tham số dòng lệnh. Module này cung cấp hai hàm và một exception để kích hoạt việc phân tích cú pháp các tham số dòng lệnh.

Phương thức getopt.getopt trong Python

Phương thức này phân tích cú pháp danh sách tham số và các tùy chọn tham số dòng lệnh Cú pháp là:

```
getopt.getopt(args, option, [long_option])
```

Chi tiết về tham số:

- **args**: Đây là danh sách tham số để được phân tích.
- **option**: Đây là chuỗi các tùy chọn mà script muốn để nhận ra. Với các tùy chọn mà yêu cầu một tham số thì nên được theo sau bởi một dấu hai chấm (:).
- **long_option**: Đây là tham số tùy ý và nếu được xác định, phải là một danh sách các chuỗi là tên các tùy chọn dài, mà được hỗ trợ. Với các tùy chọn dài yêu cầu một tham số thì nên được theo sau bởi một dấu bằng (=). Để chỉ chấp nhận các tùy chọn dài, các tùy chọn nên là một chuỗi trống.
- Phương thức này trả về trả trị bao gồm hai phần tử: phần tử đầu là một danh sách các cặp (option, value). Phần tử thứ hai là danh sách các tham số chương trình.
- Cặp option-value được trả về có một dấu gạch nối ngắn ở trước (ví dụ -x) là tùy chọn ngắn, có hai dấu gạch nối là tùy chọn dài (ví dụ --long-option).

getopt.GetoptError trong Python

Đây là một exception và nó được tạo khi thấy một tùy chọn không được nhận ra trong danh sách tham số hoặc khi một tùy chọn cần một tham số mà không cung cấp tham số nào.

Tham số cho exception là một chuỗi chỉ nguyên nhân gây ra lỗi. Các thuộc tính **msg** và **opt** cung cấp thông điệp lỗi và tùy chọn có liên quan.

Ví dụ

Giả sử bạn muốn truyền hai tên file thông qua dòng lệnh và bạn cũng muốn cung cấp một tùy chọn để kiểm tra sự sử dụng của script. Usage của script là như sau:

```
usage: test.py -i <inputfile> -o <outputfile>
```

Đây là script:

```
import sys, getopt

def main(argv):
    inputfile = ''
    outputfile = ''
    try:
        opts, args = getopt.getopt(argv,"hi:o:",["ifile=","ofile="])
    except getopt.GetoptError:
        print 'test.py -i <inputfile> -o <outputfile>'
        sys.exit(2)
    for opt, arg in opts:
        if opt == '-h':
            print 'test.py -i <inputfile> -o <outputfile>'
            sys.exit()
        elif opt in ("-i", "--ifile"):
            inputfile = arg
        elif opt in ("-o", "--ofile"):
            outputfile = arg
    print 'Input file is "', inputfile
    print 'Output file is "', outputfile

if __name__ == "__main__":
    main(sys.argv[1:])
```

Bây giờ chạy script trên như sau:

```
$ test.py -h
usage: test.py -i <inputfile> -o <outputfile>

$ test.py -i BMP -o
usage: test.py -i <inputfile> -o <outputfile>

$ test.py -i inputfile
```

```
Input file is " inputfile
```

```
Output file is "
```

Các kiểu biến trong Python

Biến là không gì khác ngoài các vị trí bộ nhớ được dành riêng để lưu trữ dữ liệu. Một khi một biến đã được lưu trữ, nghĩa là một khoảng không gian đã được cấp phát trong bộ nhớ đó.

Dựa trên kiểu dữ liệu của một biến, trình thông dịch cấp phát bộ nhớ và quyết định những gì có thể được lưu trữ trong khu nhớ dành riêng đó. Vì thế, bằng việc gán các kiểu dữ liệu khác nhau cho các biến, bạn có thể lưu trữ số nguyên, thập phân hoặc ký tự trong các biến này.

Gán các giá trị cho biến trong Python

Trong Python, chúng ta không cần khai báo biến một cách tường minh. Khi bạn gán bất cứ giá trị nào cho biến thì biến đó được khai báo một cách tự động. Phép gán được thực hiện bởi toán tử =.

Toán hạng trái của toán tử = là tên biến và toán hạng phải là giá trị được lưu trữ trong biến. Ví dụ:

```
a = 20          # Mot phép gan so nguyen
b = 100.0        # Mot so thuc
ten = "Hoang"    # Mot chuoi

print a
print b
print ten
```

Ở đây, 20, 100.0 và Hoang là các giá trị được gán cho các biến a, b và ten. Các lệnh trên sẽ cho kết quả sau:

```
20
100.0
Hoang
```

Phép đa gán (multiple assignment) trong Python

Python cho phép bạn gán một giá trị đơn cho một số biến một cách đồng thời. Python hỗ trợ hai kiểu đa gán sau:

Gán giá trị đơn cho nhiều biến, ví dụ:

```
a = b = c = 1
```

Hoặc gán nhiều giá trị cho nhiều biến, ví dụ:

```
a,b,c=5,10,15  
print a  
print b  
print c
```

Trong trường hợp này, các giá trị sẽ được gán theo thứ tự mà các biến xuất hiện.

Các kiểu dữ liệu chuẩn trong Python

Dữ liệu mà được lưu trữ trong bộ nhớ có thể có nhiều kiểu khác nhau. Ví dụ, lương của công nhân được lưu trữ dưới dạng một giá trị số còn địa chỉ của họ được lưu trữ dưới dạng các ký tự chữ-số. Python có nhiều kiểu dữ liệu chuẩn được sử dụng để xác định các hành động có thể xảy ra trên chúng và phương thức lưu trữ cho mỗi kiểu.

Python có 5 kiểu dữ liệu chuẩn là:

- Kiểu Number
- Kiểu String
- Kiểu List
- Kiểu Tuple
- Kiểu Dictionary

Ngoài kiểu Number và kiểu String mà có thể bạn đã được làm quen với các ngôn ngữ lập trình khác thì ở trong Python còn xuất hiện thêm ba kiểu dữ liệu đó là List, Tuple và Dictionary. Chúng ta sẽ tìm hiểu chi tiết từng kiểu dữ liệu trong một chương riêng (Bạn theo link để tìm hiểu chúng). Tiếp theo chúng ta tìm hiểu một số hàm đã được xây dựng sẵn trong Python để thực hiện phép chuyển đổi giữa các kiểu dữ liệu.

Chuyển đổi kiểu trong Python

Đôi khi bạn cần thực hiện một số phép chuyển đổi kiểu để thỏa mãn hàm hoặc phương thức nào đó, ... Để thực hiện điều này, đơn giản là bạn sử dụng tên kiểu như là một hàm. Dưới đây là một

số hàm đã được xây dựng sẵn để chuyển đổi từ một kiểu này sang một kiểu khác. Các hàm này trả về một đối tượng mới biểu diễn giá trị đã được chuyển đổi.

Hàm	Miêu tả
int(x [,base])	Chuyển đổi x thành một số nguyên. Tham số base xác định cơ sở nếu x là một chuỗi
long(x [,base])	Chuyển đổi x thành một long int. Tham số base xác định cơ sở nếu x là một chuỗi
float(x)	Chuyển đổi x thành một số thực
complex(real [,imag])	Chuyển đổi x thành một số phức
str(x)	Chuyển đổi x thành một chuỗi
repr(x)	Chuyển đổi đối tượng x thành một chuỗi biểu thức
eval(str)	Ước lượng một chuỗi và trả về một đối tượng
tuple(s)	Chuyển đổi s thành một Tuple
list(s)	Chuyển đổi s thành một List
set(s)	Chuyển đổi s thành một Set
dict(d)	Tạo một Dictionary. Tham số d phải là một dãy các Tuple của cặp (key, value)

frozenset(s)	Chuyển đổi s thành một Fronzen Set
chr(x)	Chuyển đổi một số nguyên thành một ký tự
unichr(x)	Chuyển đổi một số nguyên thành một ký tự Unicode
ord(x)	Chuyển đổi một ký tự đơn thành giá trị nguyên của nó
hex(x)	Chuyển đổi một số nguyên thành một chuỗi thập lục phân
oct(x)	Chuyển đổi một số nguyên thành một chuỗi bát phân

Toán tử trong Python

Toán tử là các biểu tượng cụ thể mà thực hiện một số hoạt động trên một số giá trị và cho ra một kết quả. Ví dụ biểu thức $2 + 3 = 5$, thì 2 và 3 được gọi là các toán hạng và dấu + được gọi là toán tử.

Các loại toán tử trong Python

Python hỗ trợ các loại toán tử sau:

- Toán tử số học
- Toán tử quan hệ (còn gọi là toán tử so sánh)
- Toán tử gán
- Toán tử logic
- Toán tử membership
- Toán tử identify
- Toán tử thao tác bit

Toán tử số học trong Python

Assume variable a holds 10 and variable b holds 20, then –

Toán tử	Miêu tả
//	Thực hiện phép chia, trong đó kết quả là thương số sau khi đã xóa các ký số sau dấu phẩy
+	Phép cộng
-	Phép trừ

*	Phép nhân
/	Phép chia
%	Phép chia lấy phần dư
**	Phép lấy số mũ (ví dụ $2^{**}3$ cho kết quả là 8)

Dưới đây là ví dụ minh họa các toán tử số học trong Python.

```
>>> 10+20
30
>>> 20-10
10
>>> 10*2
20
>>> 10/2
5
>>> 10%3
1
>>> 2**3
8
>>> 10//3
3
>>>
```

Toán tử quan hệ trong Python

Python hỗ trợ các toán tử quan hệ (toán tử so sánh) sau:

Toán tử	Miêu tả

<	Nhỏ hơn. Nếu giá trị của toán hạng trái là nhỏ hơn giá trị của toán hạng phải, thì điều kiện trở thành true
>	Lớn hơn
<=	Nhỏ hơn hoặc bằng
>=	Lớn hơn hoặc bằng
==	Bằng
!=	Không bằng
<>	Không bằng (tương tự !=)

Dưới đây là ví dụ minh họa cho các toán tử quan hệ trong Python:

```
>>> 10<20
True
>>> 10>20
False
>>> 10<=10
True
>>> 20>=15
True
>>> 5==6
False
>>> 5!=6
True
>>> 10<>2
True
>>>
```

Toán tử gán trong Python

Python hỗ trợ các loại toán tử gán sau:

Toán tử	Miêu tả
=	Phép gán
/=	Chia toán hạng trái cho toán hạng phải, và gán kết quả cho toán hạng trái
+=	Cộng và gán
-=	Trừ và gán
*=	Nhân và gán
%=	Chia lấy phần dư và gán
**=	Lấy số mũ và gán
//=	Thực hiện phép chia // và gán

Dưới đây là ví dụ minh họa cho các toán tử gán trong Python:

```
>>> c=10
>>> c
10
>>> c+=5
>>> c
15
>>> c-=5
>>> c
```

```
10  
  
>>> c*=2  
  
>>> c  
  
20  
  
>>> c/=2  
  
>>> c  
  
10  
  
>>> c%3  
  
>>> c  
  
1  
  
>>> c=5  
  
>>> c**=2  
  
>>> c  
  
25  
  
>>> c/=2  
  
>>> c  
  
12  
  
>>>
```

Toán tử logic trong Python

Python hỗ trợ các toán tử logic sau:

Toán tử	Miêu tả
and	Phép Và. Nếu cả hai điều kiện là true thì kết quả sẽ là true
or	Phép Hoặc. Nếu một trong hai điều kiện là true thì kết quả là true
not	Phép phủ định. Được sử dụng để đảo ngược trạng thái logic của toán hạng

Dưới đây là ví dụ minh họa cho các toán tử logic trong Python:

```
a=5>4 and 3>2
```

```
print a  
b=5>4 or 3<2  
print b  
c=not(5>4)  
print c
```

Kết quả là:

```
>>>  
True  
True  
False  
>>>
```

Toán tử thao tác bit trong Python

Toán tử thao tác bit làm việc trên các bit và thực hiện các hoạt động theo từng bit. Giả sử $a = 60$ và $b = 13$ thì định dạng nhị phân của chúng lần lượt là

$a = 0011\ 1100$ và $b = 0000\ 1101$.

Python hỗ trợ các toán tử thao tác bit sau:

Toán tử	Miêu tả	Ví dụ
$\&$	Sao chép một bit tới kết quả nếu bit này tồn tại trong cả hai toán hạng	$(a \& b) = 0000\ 1100$
$ $	Sao chép một bit tới kết quả nếu bit này tồn tại trong bất kỳ toán hạng nào	$(a b) = 61$ (tức là)

		0011 1101)
\wedge	Sao chép bit nếu nó được set (chỉ bit 1) chỉ trong một toán hạng	$(a \wedge b) = 49$ (tức là 0011 0001)
\sim	Đây là toán tử một ngôi, được sử dụng để đảo ngược bit	$(\sim a) = -61$ (tức là 1100 0011)
$<<$	Toán tử dịch trái nhị phân. Giá trị của toán hạng trái được dịch chuyển sang trái một số lượng bit đã được xác định bởi toán hạng phải	$a << = 240$ (tức là 1111 0000)
$>>$	Toán tử dịch phải nhị phân. Giá trị của toán hạng trái được dịch chuyển sang phải một số lượng bit đã được xác định bởi toán hạng phải	$a >> = 15$ (tức là 0000 1111)

Dưới đây là ví dụ minh họa cho các toán tử thao tác bit trong Python:

```
a = 60          # 60 = 0011 1100
b = 13          # 13 = 0000 1101
c = 0
```

```
c = a & b;      # 12 = 0000 1100
print "Dong 1 - Gia tri cua c la ", c

c = a | b;      # 61 = 0011 1101
print "Dong 2 - Gia tri cua c la ", c

c = a ^ b;      # 49 = 0011 0001
print "Dong 3 - Gia tri cua c la ", c

c = ~a;         # -61 = 1100 0011
print "Dong 4 - Gia tri cua c la ", c

c = a << 2;     # 240 = 1111 0000
print "Dong 5 - Gia tri cua c la ", c

c = a >> 2;     # 15 = 0000 1111
print "Dong 6 - Gia tri cua c la ", c
```

Toán tử membership trong Python

Toán tử membership trong Python kiểm tra xem biến này có nằm trong dãy (có là một trong các thành viên của dãy) hay không. Có hai toán tử membership trong Python là:

Toán tử	Miêu tả
in	Trả về true nếu một biến là nằm trong dãy các biến, nếu không là false
not in	Trả về true nếu một biến là không nằm trong dãy các biến, nếu không là false

Dưới đây là ví dụ minh họa cho các toán tử membership trong Python:

```
a=10
```

```
b=20
```

```
list=[10,20,30,40,50];
if (a in list):
    print "a la trong list da cho"
else:
    print "a la khong trong list da cho"
if(b not in list):
    print "b la khong trong list da cho"
else:
    print "b la trong list da cho"
```

Kết quả là:

```
>>>
a la trong list da cho
b la trong list da cho
>>>
```

Toán tử identify trong Python

Toán tử identify so sánh các vị trí ô nhớ của hai đối tượng. Python có hai toán tử identify là:

Toán tử	Miêu tả
is	Trả về true nếu các biến ở hai bên toán tử cùng trả về một đối tượng, nếu không là false
is not	Trả về false nếu các biến ở hai bên toán tử cùng trả về một đối tượng, nếu không là true

Dưới đây là ví dụ minh họa cho các toán tử identify trong Python:

```
a=20
b=20
if( a is b):
    print ?a,b co cung identity?
```

```
else:  
    print ?a, b la khac nhau?  
  
b=10  
  
if( a is not b):  
    print ?a,b co identity khac nhau?  
  
else:  
    print ?a,b co cung identity?
```

Kết quả là:

```
>>>  
a,b co cung identity  
a,b co identity khac nhau  
>>>
```

Thứ tự ưu tiên của các toán tử trong Python

Bạn cần chú ý thứ tự ưu tiên của các toán tử để mang lại kết quả như mong muốn trong quá trình làm việc. Bảng dưới đây liệt kê tất cả các toán tử trong Python với thứ tự ưu tiên từ cao xuống thấp.

Toán tử	Miêu tả
**	Toán tử mũ
~ + -	Phản bù; phép cộng và trừ một ngôi (với tên phương thức lần lượt là +@ và -@)
* / % //	Phép nhân, chia, lấy phần dư và phép chia //
+ -	Phép cộng và phép trừ
>> <<	Dịch bit phải và dịch bit trái

&	Phép Và Bit
^	Phép XOR và OR
<= < > >=	Các toán tử so sánh
<> == !=	Các toán tử so sánh bằng
= %= /= //=-= += *= **=	Các toán tử gán
is is not	Các toán tử Identity
in not in	Các toán tử Membership
not or and	Các toán tử logic

Điều khiển luồng trong Python

Các bạn cũng đã khá quen thuộc với các lệnh điều khiển luồng trong C, C++ như if, if else, ...
Chương này sẽ trình bày về các lệnh điều khiển luồng trong Python.

Ngôn ngữ lập trình Python coi các giá trị **khác null và khác 0 là true, và coi các giá trị là null hoặc 0 là false.**

Sau đây chúng ta cùng tìm hiểu các lệnh điều khiển luồng trong Python:

Lệnh if trong Python

Lệnh if trong Python là giống như trong ngôn ngữ C. Lệnh này được sử dụng để kiểm tra một điều kiện, nếu điều kiện là true thì lệnh của khối if sẽ được thực thi, nếu không nó sẽ bị bỏ qua.

Bạn theo link sau để tìm hiểu chi tiết về [Lệnh if trong Python](#).

Lệnh if...else trong Python

Một lệnh else có thể được sử dụng kết hợp với lệnh if. Một lệnh else chứa khối code mà thực thi nếu biểu thức điều kiện trong lệnh if được ước lượng là 0 hoặc một giá trị false. Lệnh else là lệnh tùy ý và chỉ có duy nhất một lệnh else sau lệnh if.

Bạn theo link sau để tìm hiểu chi tiết về [Lệnh if...else trong Python](#).

Lồng các lệnh if trong Python

Đôi khi có một tình huống là khi bạn muốn kiểm tra thêm một điều kiện khác sau khi một điều kiện đã được ước lượng là true. Trong tình huống như vậy, bạn có thể sử dụng các lệnh if lồng nhau trong Python.

Trong cấu trúc các lệnh if lồng nhau, bạn có thể có cấu trúc if...elif...else bên trong cấu trúc if...elif...else khác.

Bạn theo link sau để tìm hiểu chi tiết về [Lồng các lệnh if trong Python](#).

Trường hợp các Suite lệnh đơn

Bạn đã tìm hiểu về suite trong chương Cú pháp cơ bản. Nếu suite của một mệnh đề if chỉ bao gồm một dòng lệnh đơn, thì nó có thể ở trên cùng một dòng như là header của lệnh.

Dưới đây là ví dụ đơn giản về một mệnh đề if một dòng:

```
var = 6677028

if ( var == 6677028 ) : print "Gia tri cua bieu thuc la 100"

print "Beautiful!"
```

Khi code trên được thực thi sẽ cho kết quả:

```
Gia tri cua bieu thuc la 6677028
Beautiful!
```

Lệnh if trong Python

Lệnh if trong Python là giống như trong ngôn ngữ C. Lệnh này được sử dụng để kiểm tra một điều kiện, nếu điều kiện là true thì lệnh của khối if sẽ được thực thi, nếu không nó sẽ bị bỏ qua.

Cú pháp của lệnh if là:

```
if bieu_thuc:  
    cac_lenh
```

Ví dụ của lệnh if trong Python:

```
var1 = 100  
  
if var1:  
    print "1 - Nhan mot gia tri true"  
    print var1  
  
  
var2 = 0  
  
if var2:  
    print "2 - Nhan mot gia tri true"  
    print var2  
print "Good bye!"
```

Khi code trên được thực thi sẽ cho kết quả:

```
1 - Nhan mot gia tri true  
100  
Good bye!
```

Lệnh if...elif...else trong Python

Một lệnh **else** có thể được sử dụng kết hợp với lệnh **if**. Một lệnh else chứa khối code mà thực thi nếu biểu thức điều kiện trong lệnh if được ước lượng là 0 hoặc một giá trị **false**. Lệnh else là lệnh tùy ý và chỉ có **đuy nhất** một lệnh else sau lệnh if.

Cú pháp của lệnh *if...else* là:

```
if bieu_thuc:  
    cac_lenh  
  
else:  
    cac_lenh
```

Dưới đây là ví dụ minh họa lệnh if...else trong Python:

```
var1 = 100  
  
if var1:  
    print "1 - Nhan mot gia tri true"  
    print var1  
  
else:  
    print "1 - Nhan mot gia tri false"  
    print var1  
  
  
var2 = 0  
  
if var2:  
    print "2 - Nhan mot gia tri true"  
    print var2  
  
else:  
    print "2 - Nhan mot gia tri false"  
    print var2  
  
  
print "Good bye!"
```

Khi code trên được thực thi sẽ cho kết quả:

```
1 - Nhan mot gia tri true
100
2 - Nhan mot gia tri false
0
Good bye!
```

Lệnh elif trong Python

Lệnh elif cho phép bạn kiểm tra nhiều điều kiện và thực thi khối code ngay khi một trong các điều kiện được ước lượng là true. Cũng giống như lệnh else, lệnh elif là tùy ý. Tuy nhiên, không giống else mà chỉ có một lệnh được theo sau if, thì bạn có thể sử dụng **nhiều elif theo sau if**.

Cú pháp của lệnh elif là:

```
if bieu_thuc1:
    cac_lenh
elif bieu_thuc2:
    cac_lenh
elif bieu_thuc3:
    cac_lenh
else:
    cac_lenh
```

Python không cung cấp các lệnh switch hoặc case như trong các ngôn ngữ lập trình khác, tuy nhiên bạn có thể sử dụng các lệnh if...elif để thực hiện vai trò như của switch hoặc case như trong ví dụ trên.

Dưới đây là ví dụ của lệnh elif trong Python:

```
var = 100
if var == 200:
    print "1 - Nhan mot gia tri true"
    print var
elif var == 150:
    print "2 - Nhan mot gia tri true"
```

```
print var  
  
elif var == 100:  
    print "3 - Nhan mot gia tri true"  
    print var  
  
else:  
    print "4 - Nhan mot gia tri false"  
    print var  
  
print "Good bye!"
```

Khi code trên được thực thi sẽ cho kết quả:

```
3 - Nhan mot gia tri true  
100  
Good bye!
```

Lồng các lệnh if trong Python

Đôi khi có một tình huống là khi bạn muốn kiểm tra thêm một điều kiện khác sau khi một điều kiện đã được ước lượng là true. Trong tình huống như vậy, bạn có thể sử dụng các lệnh if lồng nhau trong Python.

Trong cấu trúc các lệnh if lồng nhau, bạn có thể có cấu trúc if...elif...else bên trong cấu trúc if...elif...else khác.

Cú pháp của cấu trúc lồng các lệnh if như sau:

```
if bieu_thuc1:  
    cac_lenh  
    if bieu_thuc2:  
        cac_lenh  
    elif bieu_thuc3:  
        cac_lenh  
    else:  
        cac_lenh  
    elif bieu_thuc4:  
        cac_lenh  
    else:  
        cac_lenh
```

Dưới đây là ví dụ minh họa cho cấu trúc các lệnh if lồng nhau trong Python:

```
var = 100  
  
if var < 200:  
    print "Gia tri bieu thuc la nho hon 200"  
    if var == 150:  
        print "Do la 150"  
    elif var == 100:  
        print "Do la 100"  
    elif var == 50:
```

```
print "Do la 50"  
elif var < 50:  
    print "Gia tri bieu thuc la nho hon 50"  
else:  
    print "Khong tim thay bieu thuc true"  
  
print "Good bye!"
```

Khi code trên được thực thi sẽ cho kết quả sau:

```
Gia tri bieu thuc la nho hon 200  
Do la 100  
Good bye!
```

Vòng lặp trong Python

Nói chung, các lệnh được thực thi theo các liên tiếp nhau: lệnh thứ nhất được thực thi đầu tiên và sau đó là lệnh thứ hai, thứ ba, ... Tuy nhiên có một tình huống là bạn muốn thực thi một khối code nhiều lần, thì trong trường hợp này, các ngôn ngữ lập trình cung cấp cho bạn các cấu trúc điều khiển đa dạng để làm điều này.

Bạn cũng đã khá quen thuộc với khái niệm vòng lặp trong C, hoặc C++. Ngôn ngữ Python cũng cung cấp cho bạn các kiểu vòng lặp như vòng lặp while, vòng lặp for, cấu trúc lồng vòng lặp. Bên cạnh đó, để hỗ trợ cho trình thực thi của vòng lặp, Python cũng hỗ trợ một số lệnh điều khiển vòng lặp như lệnh break, lệnh continue và lệnh pass. Phản tiếp theo chúng ta sẽ tìm hiểu sơ qua về từng loại này.

(Bạn truy cập link để tìm hiểu chi tiết.)

Các vòng lặp trong Python

Vòng lặp while: lặp đi lặp lại một lệnh hoặc một nhóm lệnh trong khi một điều kiện đã cho là TRUE. Nó kiểm tra điều kiện trước khi thực thi phần thân của vòng lặp.

Vòng lặp for: nó có khả năng lặp qua các item của bất kỳ dãy nào như một list hoặc một string.

Lồng vòng lặp: bạn có thể sử dụng một hoặc nhiều vòng lặp bên trong bất kỳ vòng lặp while, for hoặc do...while nào.

Các lệnh điều khiển vòng lặp trong Python

Các lệnh điều khiển vòng lặp thay đổi trình thực thi thông thường. Khi trình thực thi rời khỏi một phạm vi, thì tất cả các đối tượng tự động được tạo trong phạm vi đó sẽ bị hủy. Python hỗ trợ các lệnh điều khiển vòng lặp sau:

Lệnh break: kết thúc lệnh vòng lặp và truyền trình thực thi tới lệnh ngay sau vòng lặp đó.

Lệnh continue: làm cho vòng lặp nhảy qua phần còn lại của thân vòng lặp và tự động kiểm tra lại điều kiện của nó trước khi lặp lại vòng mới.

Lệnh pass: được sử dụng khi một lệnh là cần thiết theo cú pháp, nhưng bạn lại không muốn bắt cú lệnh hoặc code nào được thực thi.

Vòng lặp while trong Python

Vòng lặp while trong Python thực thi lặp đi lặp lại các lệnh hoặc phần thân của vòng lặp miễn là điều kiện đã cho là true. Khi điều kiện là false, thì điều khiển sẽ thoát ra khỏi vòng lặp. Dưới đây là cú pháp của vòng lặp while trong Python:

```
while bieu_thuc:  
    cac_lenh
```

Ở đây, cac_lenh có thể là một lệnh đơn hoặc một khối lệnh. Bieu_thuc có thể là bất kỳ biểu thức nào. Điều đáng chú ý về vòng lặp while là vòng lặp này có thể không chạy. Bởi vì khi điều kiện được kiểm tra là false, thì phần thân vòng lặp sẽ bị bỏ qua và lệnh đầu tiên ngay sau vòng lặp sẽ được thực thi.

Ví dụ minh họa cho vòng lặp while trong Python:

```
count = 0  
while (count < 9):  
    print 'So thu tu cua ban la:', count  
    count = count + 1  
  
print "Good bye!"
```

Khi code trên được thực thi sẽ cho kết quả:

```
So thu tu cua ban la: 0  
So thu tu cua ban la: 1  
So thu tu cua ban la: 2  
So thu tu cua ban la: 3  
So thu tu cua ban la: 4  
So thu tu cua ban la: 5  
So thu tu cua ban la: 6  
So thu tu cua ban la: 7  
So thu tu cua ban la: 8  
Good bye!
```

Trong ví dụ trên, gồm lệnh print và lệnh tăng, khối code sẽ được thực thi lặp đi lặp lại tới khi count không còn nhỏ hơn 9 nữa. Với mỗi lần lặp, giá trị của count được hiển thị và sau đó được tăng thêm 1.

Vòng lặp vô hạn trong Python

Một vòng lặp vô hạn là vòng lặp mà điều kiện của nó là luôn true. Bạn phải đặc biệt chú ý khi sử dụng các vòng lặp while bởi vì tồn tại khả năng là điều kiện của nó sẽ không bao giờ false, tức là làm cho vòng lặp không bao giờ kết thúc.

Một vòng lặp vô hạn có thể là rất hữu ích trong lập trình client/server, tại đó server cần chạy liên tục để mà các chương trình client có thể giao tiếp với nó khi cần thiết.

```
var = 1

while var == 1 : # Lenh nay tao mot vong lap vo han
    num = raw_input("Hay nhap mot so :")
    print "So da nhap la: ", num

print "Good bye!"
```

Khi code trên được thực thi sẽ cho kết quả:

```
Hay nhap mot so :20
So da nhap la:  20
Hay nhap mot so :29
So da nhap la:  29
Hay nhap mot so :3
So da nhap la:  3
Hay nhap mot so between :Traceback (most recent call last):
  File "test.py", line 5, in <module>
    num = raw_input("Hay nhap mot so :")
KeyboardInterrupt
```

Ví dụ trên là một vòng lặp vô hạn, và để thoát khỏi nó thì bạn cần nhấn phím CTRL+C.

Sử dụng lệnh else với vòng lặp while trong Python

Python cho phép bạn có một lệnh else được sử dụng kết hợp với một lệnh vòng lặp. Khi else được sử dụng với một vòng lặp while, thì lệnh else được thực thi khi điều kiện là false.

Ví dụ sau minh họa sự kết hợp của lệnh else với một lệnh while để in các số, miễn là số này nhỏ hơn 5, nếu không lệnh else được thực thi.

```
count = 0
while count < 5:
    print count, " la nho hon 5"
    count = count + 1
else:
    print count, " la khong nho hon 5"
```

Khi code trên được thực thi sẽ cho kết quả:

```
0 la nho hon 5
1 la nho hon 5
2 la nho hon 5
3 la nho hon 5
4 la nho hon 5
5 la khong nho hon 5
```

Suite: lệnh while trên một dòng đơn

Tương tự như cú pháp của lệnh if, nếu mệnh đề while của bạn chỉ gồm một lệnh đơn, thì nó có thể được đặt trên cùng một dòng như là header của lệnh while này, ví dụ:

```
flag = 1

while (flag): print 'Flag da cung cap nay la true!'

print "Good bye!"
```

Bạn không nên thử ví dụ này vì nó là một vòng lặp vô hạn và bạn cần nhấn CTRL+C để thoát.

Vòng lặp for trong Python

Vòng lặp for được sử dụng để lặp một biến qua một dãy (List hoặc String) theo thứ tự mà chúng xuất hiện. Sau đây là cú pháp của vòng lặp for:

```
for bien_vong_lap in day_sequense:  
    cac_lenh
```

Nếu một dãy **day_sequense** gồm một danh sách các biểu thức, nó được ước lượng đầu tiên. Sau đó, item đầu tiên trong dãy được gán cho biến vòng lặp **bien_vong_lap**. Tiếp theo, các khối lệnh **cac_lenh** được thực thi và khối lệnh này được thực thi tới khi dãy này đã được lặp xong.

Dưới đây là ví dụ minh họa vòng lặp for trong Python:

```
for letter in 'Python':      # Vi du dau tien  
    print 'Chu cai hien tai :', letter  
  
qua = ['chuoi', 'tao',  'xoai']  
for qua in qua:            # Vi du thu hai  
    print 'Ban co thich an :', qua  
  
print "Good bye!"
```

Khi code trên được thực thi sẽ cho kết quả:

```
Chu cai hien tai : P  
Chu cai hien tai : y  
Chu cai hien tai : t  
Chu cai hien tai : h  
Chu cai hien tai : o  
Chu cai hien tai : n  
Ban co thich an : chuoi  
Ban co thich an : tao  
Ban co thich an : xoai  
Good bye!
```

Lặp qua index của dãy

Một cách khác để lặp qua mỗi item là bởi chỉ mục index bên trong dãy đó. Bạn theo dõi ví dụ đơn giản sau:

```
qua = ['chuoi', 'tao', 'xoai']
for index in range(len(qua)):
    print 'Ban co thich an :', qua[index]

print "Good bye!"
```

Khi code trên được thực thi sẽ cho kết quả:

```
Ban co thich an : chuoi
Ban co thich an : tao
Ban co thich an : xoai
Good bye!
```

Ở đây, chúng ta sử dụng hàm len(), có sẵn trong Python, để cung cấp tổng số phần tử trong tuple cũng như hàm range() để cung cấp cho chúng ta dãy thực sự để lặp qua đó.

Sử dụng lệnh else với vòng lặp for trong Python

Python cho phép bạn có một lệnh else để liên hợp với một lệnh vòng lặp. Với vòng lặp for, lệnh else được thực thi khi vòng lặp đã lặp qua hết các phần tử trong list.

Ví dụ sau minh họa sự kết hợp của một lệnh else với một lệnh for để tìm kiếm các số nguyên tố từ 10 tới 20.

```
for num in range(10,20): #de lap tu 10 toi 20
    for i in range(2,num): #de lap tren cac thua so cua mot so
        if num%i == 0:      #de xac dinh thua so dau tien
            j=num/i          #de uoc luong thua so thu hai
            print '%d la bang %d * %d' % (num,i,j)
```

```
break #de di chuyen toi so tiep theo, la vong FOR dau tien  
else:           # else la mot phan cua vong lap  
    print num, 'la so nguyen to'
```

Khi code trên được thực thi sẽ cho kết quả:

```
10 la bang 2 * 5  
11 la so nguyen to  
12 la bang 2 * 6  
13 la so nguyen to  
14 la bang 2 * 7  
15 la bang 3 * 5  
16 la bang 2 * 8  
17 la so nguyen to  
18 la bang 2 * 9  
19 la so nguyen to
```

Lồng vòng lặp trong Python

Ngôn ngữ lập trình Python cho phép bạn sử dụng một vòng lặp bên trong một vòng lặp khác. Dưới đây là cú pháp và một số ví dụ để minh họa điều này.

Cú pháp lồng vòng lặp for trong Python

```
for bien_vong_lap in day_seq:  
    for bien_vong_lap in day_seq:  
        cac_lenh  
        cac_lenh
```

Cú pháp lồng vòng lặp while trong Python

```
while bieu_thuc:  
    while bieu_thuc:  
        cac_lenh  
        cac_lenh
```

Một ghi chú khác trong lồng vòng lặp là bạn có thể đặt bất cứ kiểu vòng lặp nào bên trong kiểu vòng lặp khác. Ví dụ như bạn có thể đặt một vòng lặp for bên trong một vòng lặp while hoặc ngược lại.

Ví dụ

Ví dụ sau sử dụng lồng vòng lặp để tìm số nguyên tố từ 2 tới 100.

```
i = 2  
while(i < 100):  
    j = 2  
    while(j <= (i/j)):  
        if not(i%j): break  
        j = j + 1  
    if (j > i/j) : print i, " la so nguyen to"  
    i = i + 1
```

```
print "Good bye!"
```

Khi code trên được thực thi sẽ cho kết quả:

```
2 la so nguyen to
3 la so nguyen to
5 la so nguyen to
7 la so nguyen to
11 la so nguyen to
13 la so nguyen to
17 la so nguyen to
19 la so nguyen to
23 la so nguyen to
29 la so nguyen to
31 la so nguyen to
37 la so nguyen to
41 la so nguyen to
43 la so nguyen to
47 la so nguyen to
53 la so nguyen to
59 la so nguyen to
61 la so nguyen to
67 la so nguyen to
71 la so nguyen to
73 la so nguyen to
79 la so nguyen to
83 la so nguyen to
89 la so nguyen to
97 la so nguyen to
Good bye!
```

Lệnh break trong Python

Lệnh break trong Python là giống như lệnh break trong C. Lệnh này kết thúc vòng lặp hiện tại và truyền điều khiển tới cuối vòng lặp. Lệnh break này có thể được sử dụng trong vòng lặp while và vòng lặp for. Nếu bạn đang sử dụng lồng vòng lặp, thì lệnh break kết thúc sự thực thi của vòng lặp bên trong và bắt đầu thực thi dòng code tiếp theo của khôi.

Cú pháp của lệnh break là khá đơn giản:

```
break
```

Dưới đây là ví dụ minh họa lệnh break trong Python:

```
for letter in 'Python':      # Vi du thu nhat
    if letter == 'h':
        break
    print 'Chu cai hien tai :', letter

var = 10                      # Vi du thu hai
while var > 0:
    print 'Gia tri bien hien tai la :', var
    var = var -1
    if var == 5:
        break

print "Good bye!"
```

Khi code trên được thực thi sẽ cho kết quả:

```
Chu cai hien tai : P
Chu cai hien tai : y
Chu cai hien tai : t
Gia tri bien hien tai la : 10
Gia tri bien hien tai la : 9
Gia tri bien hien tai la : 8
```

```
Gia tri bien hien tai la : 7  
Gia tri bien hien tai la : 6  
Good bye!
```

Lệnh continue trong Python

Lệnh continue trả về điều khiển tới phần ban đầu của vòng lặp. Lệnh này bỏ qua lần lặp hiện tại và bắt buộc lần lặp tiếp theo của vòng lặp diễn ra. Lệnh continue có thể được sử dụng trong vòng lặp while hoặc vòng lặp for. Dưới đây là cú pháp của lệnh continue:

```
continue
```

Dưới đây là ví dụ minh họa lệnh continue trong Python:

```
for letter in 'Python':      # Vi du thu nhat
    if letter == 'h':
        continue
    print 'Chu cai hien tai :', letter

var = 10                      # Vi du thu hai
while var > 0:
    var = var -1
    if var == 5:
        continue
    print 'Gia tri bien hien tai la :', var
print "Good bye!"
```

Khi code trên được thực thi sẽ cho kết quả:

```
Chu cai hien tai : P
Chu cai hien tai : y
Chu cai hien tai : t
Chu cai hien tai : o
Chu cai hien tai : n
Gia tri bien hien tai la : 9
Gia tri bien hien tai la : 8
Gia tri bien hien tai la : 7
Gia tri bien hien tai la : 6
Gia tri bien hien tai la : 4
```

```
Gia tri bien hien tai la : 3  
Gia tri bien hien tai la : 2  
Gia tri bien hien tai la : 1  
Gia tri bien hien tai la : 0  
Good bye!
```

Lệnh pass trong Python

Lệnh pass, giống như tên của nó, được sử dụng khi một lệnh là cần thiết theo cú pháp nhưng bạn không muốn bắt cứ lệnh hoặc khối code nào được thực thi. Lệnh pass là một hoạt động null và không có gì xảy ra khi nó thực thi. Dưới đây là cú pháp của lệnh pass:

```
pass
```

Dưới đây là ví dụ minh họa lệnh pass trong Python:

```
for letter in 'Python':  
    if letter == 'h':  
        pass  
    print 'Day la khoi pass'  
    print 'Chu cai hien tai :', letter  
  
print "Good bye!"
```

Khi code trên được thực thi sẽ cho kết quả:

```
Chu cai hien tai : P  
Chu cai hien tai : y  
Chu cai hien tai : t  
Day la khoi pass  
Chu cai hien tai : h  
Chu cai hien tai : o  
Chu cai hien tai : n  
Good bye!
```

Number trong Python

Kiểu dữ liệu Number lưu trữ các giá trị số. Chúng là các kiểu dữ liệu immutable, hay là kiểu dữ liệu không thay đổi, nghĩa là các thay đổi về giá trị của kiểu dữ liệu số này sẽ tạo ra một đối tượng được cấp phát mới.

Các đối tượng Number được tạo khi bạn gán một giá trị cho chúng. Ví dụ:

```
var1 = 1  
var2 = 10
```

Bạn cũng có thể xóa tham chiếu tới một đối tượng Number bởi sử dụng lệnh **del**. Cú pháp của lệnh **del** như sau:

```
del var1[,var2[,var3[....,varN]]]]
```

Bạn có thể xóa một đối tượng hoặc nhiều đối tượng bởi lệnh **del** này, ví dụ:

```
del var  
del var_a, var_b
```

Python hỗ trợ 4 kiểu dữ liệu số, đó là:

- **Kiểu int:** kiểu số nguyên không có dấu thập phân.
- **Kiểu long:** là các số nguyên không giới hạn kích cỡ, được sau bởi một chữ l hoặc chữ L.
- **Kiểu float:** số thực với dấu thập phân. Kiểu này cũng có thể được viết ở dạng số mũ của 10 với E hoặc e như ($2.5e2 = 2.5 \times 10^2 = 250$).
- **Kiểu số phức:** là trong dạng $a + bJ$, với a và b là số thực và J (hoặc j) biểu diễn căn bậc hai của -1. Phần thực là a và phần ảo là b. Nói chung, số phức không được sử dụng nhiều trong lập trình Python.

Ngoài ra, bạn cũng cần chú ý rằng: Python cho phép bạn sử dụng chữ l với long, nhưng chúng tôi đề nghị bạn nên sử dụng chữ hoa L để tránh rắc rối với số 1.

Chuyển đổi kiểu số trong Python

Python chuyển đổi các số một cách nội tại bên trong một biểu thức chứa các kiểu phức tạp thành một kiểu chung để ước lượng. Tuy nhiên có đôi khi bạn cần chuyển đổi tường minh một số từ kiểu này sang kiểu khác để thỏa mãn yêu cầu của một toán tử hoặc một hàm.

- Để chuyển đổi số x thành số thuần nguyên, bạn gõ **int(x)**.
- Để chuyển đổi số x thành số long, bạn gõ **long(x)**.
- Để chuyển đổi số x thành số thực, bạn gõ **float(x)**.
- Để chuyển đổi số x thành số phức với phần thực là x và phần ảo là 0, bạn gõ **complex(x)**.
- Để chuyển đổi số x và y thành số phức với phần thực là x và phần ảo là y, bạn gõ **complex(x, y)**.

Hằng toán học trong Python

Python cũng định nghĩa hai hằng toán học là: hằng toán học pi và hằng toán học e.

Để làm việc và thao tác với các số, Python cũng cung cấp cho bạn một danh sách các hàm xử lý số đa dạng. Dưới đây là danh sách các hàm.

Nhóm hàm toán học trong Python

Hàm	Miêu tả
<u>Hàm abs(x)</u>	Trị tuyệt đối của x
<u>Hàm ceil(x)</u>	Số nguyên nhỏ nhất mà không nhỏ hơn x
<u>Hàm cmp(x, y)</u>	Trả về -1 nếu x < y, trả về 0 nếu x == y, hoặc 1 nếu x > y
<u>Hàm exp(x)</u>	Trả về e^x
<u>Hàm fabs(x)</u>	Giá trị tuyệt đối của x

<u>Hàm floor(x)</u>	Số nguyên lớn nhất mà không lớn hơn x
<u>Hàm log(x)</u>	Trả về ln x, với $x > 0$
<u>Hàm log10(x)</u>	Trả về $\log_{10}(x)$, với $x > 0$.
<u>Hàm max(x1, x2,...)</u>	Trả về số lớn nhất
<u>Hàm min(x1, x2,...)</u>	Trả về số nhỏ nhất
<u>Hàm modf(x)</u>	Trả về phần nguyên và phần thập phân của x. Cả hai phần có cùng dấu với x và phần nguyên được trả về dưới dạng một số thực
<u>Hàm pow(x, y)</u>	Trả về giá trị của $x^{**}y$.
<u>Hàm round(x [,n])</u>	Làm tròn x về n ký số sau dấu thập phân. Python làm tròn theo cách sau: round(0.5) là 1.0 và round(-0.5) là -1.0
<u>Hàm sqrt(x)</u>	Trả về căn bậc hai của x, với $x > 0$

Nhóm hàm xử lý số ngẫu nhiên trong Python

Các số ngẫu nhiên được sử dụng cho các game, test, bảo mật, ... Python cung cấp các hàm sau để xử lý số ngẫu nhiên.

Hàm	Miêu tả
<u>Hàm choice(seq)</u>	Một item ngẫu nhiên trong một list, tuple, hoặc một string
<u>Hàm randrange ([start,] stop [,step])</u>	Một phần tử được lựa chọn một cách ngẫu nhiên từ dãy (start, stop, step)

<u>Hàm random()</u>	Một số thực ngẫu nhiên r trong dãy $0 \geq r > 1$
<u>Hàm seed([x])</u>	Thiết lập giá trị nguyên bắt đầu mà được sử dụng trong bộ sinh số ngẫu nhiên. Bạn nên gọi hàm này trước khi gọi bất cứ hàm ngẫu nhiên nào khác. Hàm này trả về None
<u>Hàm shuffle(lst)</u>	Sắp xếp các item trong list một cách ngẫu nhiên
<u>Hàm uniform(x, y)</u>	Một số thực ngẫu nhiên r trong dãy $x \geq r > y$

Nhóm hàm lượng giác trong Python

Dưới đây là danh sách các hàm lượng giác trong Python:

Hàm	Miêu tả
<u>Hàm acos(x)</u>	Trả về arcos của x, giá trị radian
<u>Hàm asin(x)</u>	Trả về arcsin của x, giá trị radian
<u>Hàm atan(x)</u>	Trả về arctan của x, giá trị radian
<u>Hàm atan2(y, x)</u>	Trả atan(y / x), giá trị radian
<u>Hàm cos(x)</u>	Trả về cos của x
<u>Hàm hypot(x, y)</u>	Trả về $\sqrt{x^2 + y^2}$
<u>Hàm sin(x)</u>	Trả về sin của x

<u>Hàm tan(x)</u>	Trả về tan của x
<u>Hàm degrees(x)</u>	Chuyển đổi góc x từ radian thành độ
<u>Hàm radians(x)</u>	Chuyển đổi góc x từ độ thành radian

Chuỗi (String) trong Python

String là một trong các kiểu phổ biến nhất trong Python. String trong Python là immutable. Chúng ta có thể tạo các chuỗi bằng cách bao một text trong một trích dẫn đơn hoặc trích dẫn kép. Python coi các lệnh trích dẫn đơn và kép là như nhau. Ví dụ:

```
var1 = 'Hello World!'
var2 = "Python Programming"
```

Truy cập các giá trị trong String

Python không hỗ trợ một kiểu chữ cái; chúng được coi như các chuỗi có độ dài là 1. Trong Python, String được lưu giữ dưới dạng các ký tự đơn trong vị trí ô nhớ liên tiếp nhau. Lợi thế của sử dụng String là nó có thể được truy cập từ cả hai hướng (tiến về trước forward hoặc ngược về sau backward).

Việc lập chỉ mục của cả hai hướng đều được cung cấp bởi sử dụng String trong Python:

- Chỉ mục với hướng forward bắt đầu với 0,1,2,3,...
- Chỉ mục với hướng backward bắt đầu với -1,-2,-3,...

Để truy cập các giá trị trong String, bạn sử dụng các dấu ngoặc vuông có chỉ mục ở bên trong. Ví dụ:

```
var1 = 'Hello World!'
var2 = "Python Programming"

print "var1[0]: ", var1[0]
print "var2[1:5]: ", var2[1:5]
```

Khi code trên được thực thi sẽ cho kết quả:

```
var1[0]: H
var2[1:5]: ytho
```

Cập nhật String trong Python

Bạn có thể cập nhật một chuỗi đang tồn tại bằng cách gán (hoặc tái gán) một biến cho string khác.

Giá trị mới có thể liên quan hoặc khác hoàn toàn giá trị trước đó. Ví dụ:

```
var1 = 'Hello World!'

print "Chuoi hien tai la :- ", var1[:6] + 'Python'
```

Khi code trên được thực thi sẽ cho kết quả:

```
Chuoi hien tai la :- Hello Python
```

Các ký tự thoát trong Python

Bảng dưới đây liệt kê danh sách các ký tự thoát hoặc không thể in được mà có thể được biểu diễn với dấu \.

Ký tự thoát	Biểu diễn trong hệ 16	Miêu tả
\a	0x07	Bell hoặc alert
\b	0x08	Backspace
\cx		Control-x
\C-x		Control-x
\e	0x1b	Escape
\f	0x0c	Formfeed

\M-\C-x		Meta-Control-x
\n	0x0a	Newline
\nnn		Notation trong hệ cơ số 8, ở đây n là trong dãy từ 0 tới 7
\r	0x0d	Carriage return
\s	0x20	Space
\t	0x09	Tab
\v	0x0b	Tab dọc
\x		Ký tự x
\xnn		Notation trong hệ thập lục phân, ở đây n là trong dãy từ 0..9, a..f, hoặc A..F

Các toán tử để thao tác với String trong Python

Có ba kiểu toán tử được hỗ trợ bởi String, đó là:

- Toán tử cơ bản
- Toán tử membership
- Toán tử quan hệ

Các toán tử cơ bản để thao tác với String

Có hai loại toán tử cơ bản có thể được sử dụng với String, đó là toán tử nối chuỗi + và toán tử lặp chuỗi *.

Toán tử nối chuỗi + được sử dụng để nối hai chuỗi với nhau và tạo nên một chuỗi mới. Ví dụ:

```
>>> "hoang" + "nam"
```

Sẽ cho kết quả là:

```
'hoangnam'  
>>>
```

Chú ý: Cả hai toán hạng được truyền cho phép nối chuỗi này phải cùng kiểu, nếu không sẽ tạo một lỗi. Ví dụ:

```
'abc' + 3  
>>>
```

Sẽ tạo ra một lỗi là:

```
Traceback (most recent call last):  
  File "<pyshell#5>", line 1, in <module>  
    'abc' + 3  
TypeError: cannot concatenate 'str' and 'int' objects  
>>>
```

Toán tử lặp chuỗi * sử dụng hai tham số. Một tham số là giá trị nguyên và tham số khác là chuỗi.

Toán tử lặp chuỗi này được sử dụng để lặp đi lặp lại một chuỗi một số lần nào đó. Ví dụ:

```
>>> 5*"Hoang"
```

Sẽ cho kết quả là:

```
'HoangHoangHoangHoangHoang'
```

Ghi chú: Bạn có thể sử dụng toán tử lặp chuỗi * này theo bất kỳ cách nào như int * string hoặc string * int. Cả hai tham số được truyền cho toán tử này phải không trong cùng một kiểu.

Các toán tử membership để thao tác với String

Toán tử in: trả về true nếu một ký tự là có mặt trong chuỗi đã cho, nếu không nó trả về false.

Toán tử not in: trả về true nếu một ký tự là không tồn tại trong chuỗi đã cho, nếu không nó trả về false.

Ví dụ:

```
>>> str1="javapoint"
```

```
>>> str2='sssit'  
>>> str3="seomount"  
>>> str4='java'  
>>> st5="it"  
>>> str6="seo"  
>>> str4 in str1  
True  
>>> str5 in str2  
>>> st5 in str2  
True  
>>> str6 in str3  
True  
>>> str4 not in str1  
False  
>>> str1 not in str4  
True
```

Các toán tử quan hệ để thao tác với String

Tất cả các toán tử quan hệ (như <,>, <=, >=, ==, !=, <>) cũng có thể áp dụng cho các String. Các chuỗi được so sánh dựa trên giá trị ASCII hoặc Unicode. Ví dụ:

```
>>> "HOANG"=="HOANG"  
True  
>>> "afsha">>='Afsha'  
True  
>>> "Z"<>"z"  
True
```

Giải thích: Giá trị ASCII của a là 97, b là 98 và c là 99, ... Giá trị ASCII của A là 65, B là 66, của C là 67, ... Sự so sánh giữa các chuỗi được thực hiện dựa trên giá trị ASCII.

Dấu chia chuỗi [] trong Python

Có nhiều cách để chia một chuỗi. Khi chuỗi có thể được truy cập hoặc được lập chỉ mục từ cả hai hướng forward và backward thì chuỗi cũng có thể được chia theo hai hướng này. Dưới đây là cú pháp của dấu chia chuỗi [] trong Python:

```
<ten_chuoi>[chi_muc_bat_dau:chi_muc_ket_thuc]  
hoac  
<ten_chuoi>[:chi_muc_ket_thuc]
```

```
hoac  
<ten_chuoi>[chi_muc_bat_dau:]
```

Chẳng hạn với cú pháp <ten_chuoi>[chi_muc_bat_dau:chi_muc_ket_thuc], thì toán tử này sẽ trả về các ký tự nằm trong dãy chỉ mục đã cho.

Ví dụ:

```
>>> str="Nikhil"  
>>> str[0:6]  
'Nikhil'  
>>> str[0:3]  
'Nik'  
>>> str[2:5]  
'khi'  
>>> str[:6]  
'Nikhil'  
>>> str[3:]  
'hil'
```

Ghi chú: chi_muc_bat_dau trong String là inclusive, tức là bao gồm cả ký tự tại vị trí chỉ mục đó. Còn chi_muc_ket_thuc là exclusive, tức là không bao gồm ký tự tại chỉ mục đó.

Toán tử định dạng chuỗi trong Python

Một trong những đặc điểm hay nhất trong Python là toán tử định dạng chuỗi %. Toán tử này là duy nhất cho các String và được sử dụng với hàm print(). Ví dụ:

```
print "Ten toi la %s va toi nang %d kg!" % ('Hoang', 71)
```

Khi code trên được thực thi sẽ cho kết quả:

```
Ten toi la Hoang va toi nang 71 kg!
```

Bảng dưới đây liệt kê danh sách đầy đủ các biểu tượng có thể được sử dụng với toán tử %:

Biểu tượng định dạng	Chuyển đổi
----------------------	------------

%C	Ký tự
%s	Chuyển đổi thành chuỗi thông qua hàm str() trước khi định dạng
%i	Số nguyên thập phân có dấu
%d	Số nguyên thập phân có dấu
%u	Số nguyên thập phân không dấu
%o	Số nguyên hệ bát phân
%x	Số nguyên hệ thập lục phân (các chữ cái thường)
%X	Số nguyên hệ thập lục phân (các chữ cái hoa)
%e	Ký hiệu số mũ (với chữ thường 'e')
%E	Ký hiệu số mũ (với chữ hoa 'E')
%f	Số thực dấu chấm động
%g	Viết gọn của %f và %e
%G	Viết gọn của %f và %E

Trích dẫn tam (triple quote) trong Python

Trích dẫn tam trong Python cho phép các chuỗi có thể trải rộng trên nhiều dòng, bao gồm đúng nguyên văn của các newline, tab và bất kỳ ký tự đặc biệt nào khác. Bạn theo dõi đoạn code sau:

```
para_str = """day la mot chuoi day gom nhieu dong  
va gom mot so ky tu khong in duoc chang han nhu  
TAB ( \t ) chung se duoc hien thi dung nguyen van nhu the."""  
print para_str
```

Khi code trên được thực thi, nó cho kết quả như dưới đây. Bạn chú ý cách mỗi ký tự đặc biệt đã được chuyển đổi thành dạng được in của nó.

```
day la mot chuoi day gom nhieu dong  
va gom mot so ky tu khong in duoc chang han nhu  
TAB ( \t ) chung se duoc hien thi dung nguyen van nhu the.
```

Các chuỗi thô (raw string) không coi dấu \ như là một ký tự đặc biệt. Mỗi ký tự bạn đặt vào trong một chuỗi thô sẽ tồn tại giống như cách bạn đã viết nó.

Để hiểu rõ vấn đề này, trước hết bạn theo dõi ví dụ:

```
print 'C:\\nowhere'
```

Khi code trên được thực thi sẽ cho kết quả:

```
C:\\nowhere
```

Bây giờ sử dụng chuỗi thô. Chúng ta đã đặt biểu thức trong r'bieu_thuc' như sau:

```
print r'C:\\nowhere'
```

Khi code trên được thực thi sẽ cho kết quả:

```
C:\\nowhere
```

Chuỗi dạng Unicode trong Python

Các chuỗi thông thường trong Python được lưu trữ nội tại dưới dạng ASCII 8 bit, trong khi các chuỗi Unicode được lưu trữ dưới dạng Unicode 16 bit. Điều này cho phép để có một tập hợp các

ký tự đa dạng hơn, bao gồm các ký tự đặc biệt từ hầu hết các ngôn ngữ trên thế giới. Bạn theo dõi ví dụ:

```
print u'Hello, world! '
```

Khi code trên được thực thi sẽ cho kết quả:

```
Hello, world!
```

Như bạn có thể thấy, các chuỗi dạng Unicode sử dụng tiền tố u, trong khi các chuỗi thường sử dụng tiền tố r.

Các phương thức và hàm đã xây dựng sẵn để xử lý chuỗi trong Python

Python cung cấp các phương thức đa dạng đã được xây dựng sẵn để thao tác với các chuỗi. Bảng dưới đây liệt kê các phương thức này. Bạn truy cập link để thấy ví dụ chi tiết.

STT	Phương thức và Miêu tả
1	<u>Phương thức capitalize()</u> Viết hoa chữ cái đầu tiên của chuỗi
2	<u>Phương thức center(width, fillchar)</u> Trả về một chuỗi mới, trong đó chuỗi ban đầu đã được cho vào trung tâm và hai bên đó là các fillchar sao cho tổng số ký tự của chuỗi mới là width
3	<u>Phương thức count(str, beg= 0,end=len(string))</u> Đếm xem chuỗi str này xuất hiện bao nhiêu lần trong chuỗi string hoặc chuỗi con của string nếu bạn cung cấp chỉ mục ban đầu start và chỉ mục kết thúc end

4	<u>Phương thức decode(encoding='UTF-8',errors='strict')</u> Giải mã chuỗi bởi sử dụng encoding đã cho
5	<u>Phương thức encode(encoding='UTF-8',errors='strict')</u> Trả về phiên bản chuỗi đã được mã hóa của chuỗi ban đầu. Nếu có lỗi xảy ra, thì chương trình sẽ tạo một ValueError trừ khi các lỗi này được cung cấp với ignore hoặc replace
6	<u>Phương thức endswith(suffix, beg=0, end=len(string))</u> Xác định xem nếu chuỗi string hoặc chuỗi con đã cho của string (nếu bạn cung cấp chỉ mục bắt đầu beg và chỉ mục kết thúc end) kết thúc với hậu tố suffix thì trả về true, nếu không thì phương thức này trả về false
7	<u>Phương thức expandtabs(tabsize=8)</u> Mở rộng các tab trong chuỗi tới số khoảng trắng đã cho; mặc định là 8 space cho mỗi tab nếu bạn không cung cấp tabsize
8	<u>Phương thức find(str, beg=0 end=len(string))</u> Xác định xem chuỗi str có xuất hiện trong chuỗi string hoặc chuỗi con đã cho của string (nếu bạn cung cấp chỉ mục bắt đầu beg và chỉ mục kết thúc end), nếu xuất hiện thì trả về chỉ mục của str, còn không thì trả về -1
9	<u>Phương thức index(str, beg=0, end=len(string))</u> Tương tự như find(), nhưng tạo ra một ngoại lệ nếu str là không được tìm thấy
10	<u>Phương thức isalnum()</u> Trả về true nếu chuỗi có ít nhất một ký tự và tất cả ký tự là chữ-số. Nếu không hàm sẽ

	trả về false
11	<u>Phương thức isalpha()</u> Trả về true nếu chuỗi có ít nhất 1 ký tự và tất cả ký tự là chữ cái. Nếu không phương thức sẽ trả về false
12	<u>Phương thức isdigit()</u> Trả về true nếu chuỗi chỉ chứa các ký số, nếu không là false
13	<u>Phương thức islower()</u> Trả về true nếu tất cả ký tự trong chuỗi là ở dạng chữ thường, nếu không là false
14	<u>Phương thức isnumeric()</u> Trả về true nếu một chuỗi dạng Unicode chỉ chứa các ký tự số, nếu không là false
15	<u>Phương thức isspace()</u> Trả về true nếu chuỗi chỉ chứa các ký tự khoảng trắng whitespace, nếu không là false
16	<u>Phương thức istitle()</u> Trả về true nếu chuỗi là ở dạng titlecase, nếu không là false
17	<u>Phương thức isupper()</u> Trả về true nếu tất cả ký tự trong chuỗi là chữ hoa

18	<u>Phương thức join(seq)</u> Nối chuỗi các biểu diễn chuỗi của các phần tử trong dãy seq thành một chuỗi
19	<u>Phương thức len(string)</u> Trả về độ dài của chuỗi
20	<u>Phương thức ljust(width[, fillchar])</u> Trả về một chuỗi mới, trong đó có chuỗi ban đầu được căn chỉnh vào bên trái và bên phải là các fillchar sao cho tổng số ký tự là width
21	<u>Phương thức lower()</u> Chuyển đổi tất cả chữ hoa trong chuỗi sang kiểu chữ thường
22	<u>Phương thức lstrip()</u> Xóa tất cả các khoảng trắng ban đầu (leading) trong chuỗi
23	<u>Phương thức maketrans()</u> Trả về một bảng thông dịch được sử dụng trong hàm translate
24	<u>Phương thức max(str)</u> Trả về ký tự chữ cái lớn nhất từ chuỗi str đã cho
25	<u>Phương thức min(str)</u>

	Trả về ký tự chữ cái nhỏ nhất từ chuỗi str đã cho
26	<u>Phương thức replace(old, new [, max])</u> Thay thế tất cả sự xuất hiện của old trong chuỗi với new với số lần xuất hiện max (nếu cung cấp)
27	<u>Phương thức rfind(str, beg=0,end=len(string))</u> Tương tự hàm find(), nhưng trả về chỉ mục cuối cùng
28	<u>Phương thức rindex(str, beg=0, end=len(string))</u> Giống index(), nhưng trả về chỉ mục cuối cùng nếu tìm thấy
29	<u>Phương thức rjust(width[, fillchar])</u> Trả về một chuỗi mới, trong đó có chuỗi ban đầu được căn chỉnh vào bên phải và bên trái là các fillchar sao cho tổng số ký tự là width
30	<u>Phương thức rstrip()</u> Xóa bỏ tất cả các khoảng trắng trắng ở cuối (trailing) của chuỗi
31	<u>Phương thức split(str="", num=string.count(str))</u> Chia chuỗi theo delimiter đã cho (là space nếu không được cung cấp) và trả về danh sách các chuỗi con; nếu bạn cung cấp num thì chia chuỗi thành num chuỗi con
32	<u>Phương thức splitlines(num=string.count("\n"))</u> Trả về một List gồm tất cả các dòng trong chuỗi, và tùy ý xác định các ngắt dòng (nếu

	num được cung cấp và là true).
33	<u>Phương thức startswith(str, beg=0,end=len(string))</u> Xác định xem chuỗi hoặc chuỗi con (nếu bạn cung cấp chỉ mục bắt đầu beg và chỉ mục kết thúc end) có bắt đầu với chuỗi con str không, nếu có trả về true, nếu không là false
34	<u>Phương thức strip([chars])</u> Thực hiện cả hai phương thức lstrip() và rstrip() trên chuỗi
35	<u>Phương thức swapcase()</u> Đảo ngược kiểu của tất cả ký tự trong chuỗi
36	<u>Phương thức title()</u> Trả về một bản sao của chuỗi trong đó tất cả ký tự đầu tiên của tất cả các từ là ở kiểu chữ hoa.
37	<u>Phương thức translate(table, deletechars="")</u> Trả về một bản sao đã được thông dịch của chuỗi
38	<u>Phương thức upper()</u> Chuyển đổi các chữ thường trong chuỗi thành chữ hoa
39	<u>Phương thức zfill (width)</u> Trả về một chuỗi mới, trong đó bao gồm chuỗi ban đầu và được đệm thêm với các số 0 vào bên trái sao cho tổng ký tự là width

40

Phương thức isdecimal()

Trả về true nếu một chuỗi dạng Unicode chỉ chứa các ký tự thập phân, nếu không là false

List trong Python

Cấu trúc dữ liệu cơ bản nhất trong Python là dãy (sequence). Mỗi phần tử trong dãy được gán một số, là vị trí hoặc chỉ mục của nó. Chỉ mục đầu tiên là 0, chỉ mục thứ hai là 1, và ...

Python có sáu kiểu dãy đã được xây dựng sẵn, và trong loạt bài này chúng ta sẽ tìm hiểu hai kiểu được sử dụng phổ biến nhất là List và Tuple.

List trong Python

List trong Python là cấu trúc dữ liệu mà có khả năng lưu giữ các kiểu dữ liệu khác nhau.

List trong Python là thay đổi (mutable), nghĩa là Python sẽ không tạo một List mới nếu bạn sửa đổi một phần tử trong List.

List là một container mà giữ các đối tượng khác nhau trong một thứ tự đã cho. Các hoạt động khác nhau như chèn hoặc xóa có thể được thực hiện trên List.

Một List có thể được tạo ra bởi lưu trữ một dãy các kiểu giá trị khác nhau được phân biệt bởi các dấu phẩy. Dưới đây là cú pháp để tạo List:

```
<ten_list>=[giatri1, giatri2, ..., giatriN];
```

Ví dụ:

```
list1 = ['vatly', 'hoahoc', 1997, 2000];
list2 = [1, 2, 3, 4, 5];
list3 = ["a", "b", "c", "d"];
```

Một List trong Python được bao xung quanh bởi các dấu ngoặc vuông [].

Tương tự như chỉ mục của chuỗi, chỉ mục của List bắt đầu từ 0.

Truy cập các giá trị trong List trong Python

Để truy cập các giá trị trong List, bạn sử dụng cú pháp sau:

```
<ten_list>[index]
```

để lấy giá trị có sẵn tại chỉ mục đó.

Ví dụ:

```
list1 = ['vatly', 'hoahoc', 1997, 2000];
list2 = [1, 2, 3, 4, 5, 6, 7];

print "list1[0]: ", list1[0]
print "list2[1:5]: ", list2[1:5]
```

Khi code trên được thực thi sẽ cho kết quả:

```
list1[0]: vatly
list2[1:5]: [2, 3, 4, 5]
```

Ghi chú: Trình tổ chức bộ nhớ nội tại

List không lưu trữ các phần tử một cách trực tiếp tại chỉ mục. Sự thực là một tham chiếu được lưu trữ tại mỗi chỉ mục mà tham chiếu tới đối tượng được lưu trữ ở đâu đó trong bộ nhớ. Điều này là do một số đối tượng có thể lớn hơn một số đối tượng khác và vì thế chúng được lưu trữ tại một vị trí bộ nhớ khác.

Các hoạt động cơ bản trên List trong Python

Bạn có thể thực hiện các hoạt động nối với toán tử + hoặc hoạt động lặp với * như trong các chuỗi. Điểm khác biệt là ở đây nó tạo một List mới, không phải là một chuỗi.

Ví dụ cho nối List:

```
list1=[10,20]
       list2=[30,40]
       list3=list1+list2
       print list3
```

Kết quả là:

```
>>>
[10, 20, 30, 40]
>>>
```

Ghi chus: Toán tử + ngũ ý rằng cả hai toán hạng được truyền cho nó phải là List, nếu không sẽ cho một lỗi như ví dụ sau:

```
list1=[10,20]
list1+30
print list1
```

Kết quả là:

```
Traceback (most recent call last):
      File "C:/Python27/lis.py", line 2, in <module>
        list1+30
```

Ví dụ cho lặp List:

```
list1=[10,20]
print list1*1
```

Kết quả là:

```
>>>
[10, 20]
>>>
```

Cập nhật List trong Python

Bạn có thể cập nhật một hoặc nhiều phần tử của List bởi gán giá trị cho chỉ mục cụ thể đó. Cú pháp:

```
<ten_list>[index]=<giatri>
```

Ví dụ:

```
list = ['vatly', 'hoahoc', 1997, 2000];

print "Gia tri co san tai chi muc thu 2 : "
print list[2]
list[2] = 2001;
print "Gia tri moi tai chi muc thu 2 : "
```

```
print list[2]
```

Khi code trên được thực thi sẽ cho kết quả:

```
Gia tri co san tai chi muc thu 2 :  
1997  
Gia tri moi tai chi muc thu 2 :  
2001
```

Phụ thêm phần tử vào cuối một List

Phương thức `append()` được sử dụng để phụ thêm phần tử vào cuối một List. Cú pháp:

```
<ten_list>.append(item)
```

Ví dụ:

```
list1=[10, "hoang", 'z']  
print "Cac phan tu cua List la: "  
print list1  
list1.append(10.45)  
print "Cac phan tu cua List sau khi phu them la: "  
print list1
```

Khi code trên được thực thi sẽ cho kết quả:

```
>>>  
Cac phan tu cua List la:  
[10, 'hoang', 'z']  
Cac phan tu cua List sau khi phu them la:  
[10, 'hoang', 'z', 10.45]  
>>>
```

Xóa phần tử trong List

Để xóa một phần tử trong List, bạn có thể sử dụng lệnh `del` nếu bạn biết chính xác phần tử nào bạn muốn xóa hoặc sử dụng phương thức `remove()` nếu bạn không biết. Ví dụ:

```
list1 = ['vatly', 'hoahoc', 1997, 2000];
```

```
print list1
del list1[2];
print "Cac phan tu cua List sau khi xoa gia tri tai chi muc 2 : "
print list1
```

Khi code trên được thực thi sẽ cho kết quả:

```
['vatly', 'hoahoc', 1997, 2000]
Cac phan tu cua List sau khi xoa gia tri tai chi muc 2 :
['vatly', 'hoahoc', 2000]
```

Bạn cũng có thể sử dụng del để xóa tất cả phần tử từ chi_muc_bat_dau tới chi_muc_ket_thuc như sau:

```
list1=[10, 'hoang', 50.8, 'a', 20, 30]
print list1
del list1[0]
print list1
del list1[0:3]
print list1
```

Khi code trên được thực thi sẽ cho kết quả:

```
>>>
[10, 'hoang', 50.8, 'a', 20, 30]
['hoang', 50.8, 'a', 20, 30]
[20, 30]
>>>
```

Các hàm và phương thức đã xây dựng sẵn để xử lý List trong Python

Ngoài các phương thức kể trên, Python còn xây dựng sẵn rất nhiều hàm và phương thức để bạn có thể sử dụng khi làm việc với List. Bảng dưới đây liệt kê các phương thức này. Bạn truy cập link để thấy ví dụ chi tiết.

Danh sách các hàm xử lý List trong Python:

STT	Hàm và Miêu tả
1	<u>Hàm cmp(list1, list2)</u> So sánh các phần tử trong cả hai list
2	<u>Hàm len(list)</u> Trả về độ dài của list
3	<u>Hàm max(list)</u> Trả về phần tử có giá trị lớn nhất trong list
4	<u>Hàm min(list)</u> Trả về phần tử có giá trị nhỏ nhất trong list
5	<u>Hàm list(seq)</u> Chuyển đổi một tuple thành list

Danh sách các phương thức xử lý List trong Python:

STT	Phương thức và Miêu tả
1	<u>Phương thức list.append(obj)</u> Phụ thêm đối tượng obj vào cuối list
2	<u>Phương thức list.count(obj)</u>

	Đếm xem có bao nhiêu lần mà obj xuất hiện trong list
3	<u>Phương thức list.extend(seq)</u> Phụ thêm các nội dung của seq vào cuối list
4	<u>Phương thức list.index(obj)</u> Trả về chỉ mục thấp nhất trong list mà tại đó obj xuất hiện
5	<u>Phương thức list.insert(index, obj)</u> Chèn đối tượng obj vào trong list tại index đã cho
6	<u>Phương thức list.pop(obj=list[-1])</u> Xóa và trả về phần tử cuối cùng hoặc đối tượng obj có chỉ mục đã cung cấp từ list đã cho
7	<u>Phương thức list.remove(obj)</u> Xóa đối tượng obj từ list
8	<u>Phương thức list.reverse()</u> Đảo ngược thứ tự các đối tượng trong list
9	<u>Phương thức list.sort([func])</u> Sắp xếp các đối tượng của list, sử dụng hàm so sánh nếu được cung cấp

Tuple trong Python

Một tuple là một dãy các đối tượng không thay đổi (immutable) trong Python, vì thế tuple không thể bị thay đổi. Các tuple cũng là các dãy giống như List.

Không giống List mà sử dụng các dấu ngoặc vuông, thì tuple sử dụng các dấu ngoặc đơn. Các đối tượng trong tuple được phân biệt bởi dấu phẩy và được bao quanh bởi dấu ngoặc đơn (). Giống như chỉ mục của chuỗi, chỉ mục của tuple bắt đầu từ 0.

Ví dụ:

```
>>> data=(10,20,'ram',56.8)
>>> data2="a",10,20.9
>>> data
(10, 20, 'ram', 56.8)
>>> data2
('a', 10, 20.9)
>>>
```

Ghi chú: Nếu các dấu ngoặc đơn không được cung cấp với một dãy, thì nó được coi như là tuple.

Một tuple trống không chứa phần tử nào, ví dụ:

```
tup1 = ();
```

Với một tuple chỉ có một giá trị đơn, thì phải có một dấu phẩy ở cuối, ví dụ:

```
tup1 = (50,);
```

Các tuple cũng có thể được lồng vào nhau, ví dụ:

```
tup11='a','hoang',10.56
tup12=tup11,(10,20,30)
print tup11
print tup12
```

Kết quả:

```
>>>
```

```
('a', 'hoang', 10.56)
 (('a', 'hoang', 10.56), (10, 20, 30))
>>>
```

Truy cập các giá trị trong tuple trong Python

Để truy cập các giá trị trong tuple, bạn sử dụng cách tương tự như khi truy cập các phần tử trong List. Ví dụ:

```
tup1 = ('vatly', 'hoahoc', 1997, 2000);
tup2 = (1, 2, 3, 4, 5, 6, 7 );

print "tup1[0]: ", tup1[0]
print "tup2[1:5]: ", tup2[1:5]
```

Khi code trên được thực thi sẽ cho kết quả:

```
tup1[0]: vatly
tup2[1:5]: [2, 3, 4, 5]
```

Các hoạt động cơ bản trên tuple trong Python

Giống như String và List, bạn cũng có thể sử dụng toán tử nối + và toán tử lặp * với tuple. Điểm khác biệt là nó tạo ra một tuple mới, không tạo ra một chuỗi hay list.

Ví dụ cho toán tử +:

```
data1=(1,2,3,4)
data2=('x','y','z')
data3=data1+data2
print data1
print data2
print data3
```

Kết quả là:

```
>>>
(1, 2, 3, 4)
```

```
('x', 'y', 'z')
(1, 2, 3, 4, 'x', 'y', 'z')
>>>
```

Ghi chú: Dãy mới được tạo là một Tuple mới.

Ví dụ cho toán tử +:

```
tuple1=(10,20,30);
tuple2=(40,50,60);
print tuple1*2
print tuple2*3
```

Kết quả là:

```
>>>
(10, 20, 30, 10, 20, 30)
(40, 50, 60, 40, 50, 60, 40, 50, 60)
>>>
```

Xóa các phần tử của tuple trong Python

Xóa các phần tử đơn của tuple là điều không thể. Bạn chỉ có thể xóa toàn bộ tuple với lệnh del như ví dụ sau:

```
data=(10,20,'hoang',40.6,'z')
print data
del data      #se xoa du lieu cua tuple
print data      #se hien thi mot error boi vi tuple da bi xoa
```

Code trên sẽ cho kết quả sau. Bạn chú ý rằng sẽ có một exception được tạo ra, đó là bởi vì sau khi xóa tup thì tuple này không tồn tại nữa.

```
>>>
(10, 20, 'hoang', 40.6, 'z')
Traceback (most recent call last):
  File "C:/Python27/t.py", line 4, in >module<
    print data
NameError: name 'data' is not defined
>>>
```

Cập nhật phần tử trong tuple trong Python

Các phần tử của Tuple không thể được cập nhật. Đó là bởi vì tuple là không thay đổi (immutable).

Tuy nhiên, các tuple có thể được sử dụng để tạo nên một tuple mới.

Ví dụ sau sẽ tạo một exception:

```
data=(10,20,30)
data[0]=100
print data
```

Khi code trên được thực thi sẽ cho kết quả:

```
>>>
Traceback (most recent call last):
  File "C:/Python27/t.py", line 2, in >module<
    data[0]=100
TypeError: 'tuple' object does not support item assignment
>>>
```

Ví dụ tạo một tuple mới từ các tuple đang tồn tại:

```
data1=(10,20,30)
data2=(40,50,60)
data3=data1+data2
print data3
```

Khi code trên được thực thi sẽ cho kết quả:

```
>>>
(10, 20, 30, 40, 50, 60)
>>>
```

Tập hợp đối tượng mà không có dấu giới hạn

Bất kỳ tập hợp nào gồm nhiều đối tượng, được phân biệt bởi dấu phẩy, được viết mà không có các biểu tượng nhận diện (chẳng hạn như dấu ngoặc vuông cho List, dấu ngoặc đơn cho Tuple, ...) thì Python mặc định chúng là Tuple. Ví dụ:

```
print 'abc', -4.24e93, 18+6.6j, 'xyz'  
x, y = 1, 2;  
print "Gia tri cua x , y : ", x,y
```

Khi code trên được thực thi sẽ cho kết quả:

```
abc -4.24e+93 (18+6.6j) xyz  
Gia tri cua x , y : 1 2
```

Các hàm được xây dựng sẵn cho tuple trong Python

Bảng dưới liệt kê các hàm đã được xây dựng sẵn để thao tác với Tuple trong Python, bạn theo link để tìm hiểu chi tiết:

STT	Hàm và Miêu tả
1	<u>Hàm cmp(tuple1, tuple2)</u> So sánh hai tuple với nhau
2	<u>Hàm len(tuple)</u> Trả về độ dài của tuple
3	<u>Hàm max(tuple)</u> Trả về item có giá trị lớn nhất từ một tuple đã cho
4	<u>Hàm min(tuple)</u> Trả về item có giá trị nhỏ nhất từ một tuple đã cho
5	<u>Hàm tuple(seq)</u>

Chuyển đổi một dãy thành tuple

Tại sao chúng ta sử dụng tuple?

- Trình xử lý các tuple là nhanh hơn các List.
- Làm cho dữ liệu an toàn hơn bởi vì tuple là không thay đổi (immutable) và vì thế nó không thể bị thay đổi.
- Các tuple được sử dụng để định dạng String.

Dictionary trong Python

Dictionary trong Python là một tập hợp các cặp key và value chưa qua sắp xếp. Nó là một container mà chứa dữ liệu, được bao quanh bởi các dấu ngoặc mỏng đơn {}. Mỗi cặp key-value được xem như là một item. Key mà đã truyền cho item đó phải là duy nhất, trong khi đó value có thể là bất kỳ kiểu giá trị nào. Key phải là một kiểu dữ liệu không thay đổi (immutable) như chuỗi, số hoặc tuple.

Key và value được phân biệt riêng rẽ bởi một dấu hai chấm (:). Các item phân biệt nhau bởi một dấu phẩy (,). Các item khác nhau được bao quanh bên trong một cặp dấu ngoặc mỏng đơn tạo nên một Dictionary trong Python

Ví dụ:

```
data={100:'Hoang' ,101:'Nam' ,102:'Binh'}  
print data
```

Kết quả là:

```
>>>  
{100: 'Hoang', 101: 'Nam', 102: 'Binh'}  
>>>
```

Các thuộc tính của key trong Dictionary

Không có hạn chế nào với các value trong Dictionary, tuy nhiên với key thì bạn cần chú ý các điểm sau:

(a) Nhiều hơn một entry cho mỗi key là không được phép. Nghĩa là không cho phép bắn sao các key được xuất hiện. Khi bắt gặp nhiều bắn sao key trong phép gán, thì phép gán cuối cùng được thực hiện. Ví dụ:

```
dict = {'Ten': 'Hoang', 'Tuoi': 7, 'Ten': 'Nam'};  
  
print "dict['Ten']: ", dict['Ten']
```

Kết quả là:

```
dict['Ten']: Nam
```

(b) Key phải là immutable. Nghĩa là bạn chỉ có thể sử dụng chuỗi, số hoặc tuple làm key của Dictionary. Dưới đây là ví dụ đơn giản:

```
dict = {[ 'Ten' ]: 'Hoang', 'Tuoi': 7};

print "dict['Ten']: ", dict['Ten']
```

Khi code trên được thực thi sẽ cho kết quả:

```
Traceback (most recent call last):
  File "test.py", line 3, in <module>
    dict = {[ 'Ten' ]: 'Hoang', 'Tuoi': 7};
TypeError: list objects are unhashable
```

Truy cập các giá trị trong Dictionary trong Python

Khi chỉ mục không được định nghĩa với Dictionary, thì các giá trị trong Dictionary có thể được truy cập thông qua các key của chúng. Cú pháp:

```
<ten_dictionary>[key]
```

Ví dụ:

```
data1={'Id':100, 'Ten':'Thanh', 'Nghenghiep':'Developer'}
data2={'Id':101, 'Ten':'Chinh', 'Nghenghiep':'Trainer'}
print "Id cua nhan vien dau tien la:",data1['Id']
print "Id cua nhan vien thu hai la:",data2['Id']
print "Ten cua nhan vien dau tien la:",data1['Ten']
print "Nghe nghiep cua nhan vien thu hai la:",data2['Nghenghiep']
```

Kết quả là:

```
>>>
Id cua nhan vien dau tien la 100
Id cua nhan vien thu hai la 101
Ten cua nhan vien dau tien la is Thanh
```

```
Nghe nghiep cua nhan vien thu hai la Trainer
```

```
>>>
```

Nếu bạn cố gắng truy cập một item với một key nào mà không là một phần của Dictionary nào, thì bạn sẽ nhận một lỗi như sau:

```
dict = {'Ten': 'Hoang', 'Tuoi': 7, 'Lop': 'Lop1'};  
  
print "dict['Huong']: ", dict['Huong']
```

Code trên sẽ cho một lỗi là:

```
dict['Hoang']:  
Traceback (most recent call last):  
  File "test.py", line 4, in <module>  
    print "dict['Huong']: ", dict['Huong'];  
KeyError: 'Huong'
```

Cập nhật Dictionary trong Python

Item (cặp key-value) có thể được cập nhật. Bạn cập nhật một Dictionary bằng cách thêm một entry mới hoặc một cặp key-value mới, sửa đổi một entry đã tồn tại, hoặc xóa một entry đang tồn tại như trong ví dụ đơn giản sau:

```
data1={'Id':100, 'Ten':'Thanh', 'Nghenghiep':'Developer'}  
data2={'Id':101, 'Ten':'Chinh', 'Nghenghiep':'Trainer'}  
data1['Nghenghiep']='Manager'  
data2['Mucluong']=17000000  
data1['Mucluong']=12000000  
print data1  
print data2
```

Khi code trên được thực thi sẽ cho kết quả:

```
>>>  
{'Mucluong': 12000000, 'Nghenghiep': 'Manager', 'Id': 100, 'Ten': 'Thanh'}  
{'Mucluong': 17000000, 'Nghenghiep': 'Trainer', 'Id': 101, 'Ten': 'Chinh'}
```

>>>

Xóa phần tử từ Dictionary trong Python

Với Dictionary, bạn có thể xóa một phần tử đơn hoặc xóa toàn bộ nội dung của Dictionary đó. Bạn sử dụng lệnh del để thực hiện các hoạt động này.

Cú pháp để xóa một item từ Dictionary:

```
del ten_dictionary[key]
```

Để xóa cả Dictionary, bạn sử dụng cú pháp:

```
del ten_dictionary
```

Ví dụ:

```
data={100:'Hoang', 101:'Thanh', 102:'Nam'}  
del data[102]  
print data  
del data  
print data #se hien thi mot error boi vi Dictionary da bi xoa.
```

Code trên sẽ cho kết quả như dưới đây. Bạn có thể thấy một ngoại lệ được tạo ra bởi vì sau khi xóa data thì Dictionary này không tồn tại nữa.

```
>>>  
{100: 'Hoang', 101: 'Thanh'}  
  
Traceback (most recent call last):  
  File "C:/Python27/dict.py", line 5, in >module<  
    print data  
NameError: name 'data' is not defined  
>>>
```

Các hàm và phương thức đã được xây dựng sẵn cho Dictionary trong Python

Python đã xây dựng sẵn các hàm sau để được sử dụng với Dictionary. Bạn có thể theo dõi ví dụ chi tiết về các hàm này ở phần dưới đây.

STT	Hàm và Miêu tả
1	<u>Hàm cmp(dict1, dict2)</u> So sánh các phần tử của cả hai dict
2	<u>Hàm len(dict)</u> Độ dài của dict. Nó sẽ là số item trong Dictionary này
3	<u>Hàm str(dict)</u> Tạo ra một biểu diễn chuỗi có thể in được của một dict
4	<u>Hàm type(variable)</u> Trả về kiểu của biến đã truyền. Nếu biến đã truyền là Dictionary, thì nó sẽ trả về một kiểu Dictionary

Các phương thức đã được xây dựng sẵn cho Dictionary trong Python:

STT	Phương thức và Miêu tả
1	<u>Phương thức dict.clear()</u> Xóa tất cả phần tử của dict
2	<u>Phương thức dict.copy()</u> Trả về bản sao của dict
3	<u>Phương thức fromkeys(seq,value1)/ fromkeys(seq)</u> Được sử dụng để tạo một Dictionary mới từ dãy seq và value1. Trong đó dãy seq tạo

	nên các key và tất cả các key chia sẻ các giá trị từ value1. Trong trường hợp value1 không được cung cấp thì value của các key được thiết lập là None
4	<u>Phương thức dict.get(key, default=None)</u> Trả về giá trị của key đã cho. Nếu key không có mặt thì phương thức này trả về None
5	<u>Phương thức dict.has_key(key)</u> Trả về true nếu key là có mặt trong Dictionary, nếu không là false
6	<u>Phương thức dict.items()</u> Trả về tất cả các cặp (key-value) của một Dictionary
7	<u>Phương thức dict.keys()</u> Trả về tất cả các key của một Dictionary
8	<u>Phương thức dict.setdefault(key, default=None)</u> Tương tự get(), nhưng sẽ thiết lập dict[key]=default nếu key là không tồn tại trong dict
9	<u>Phương thức dict.update(dict2)</u> Được sử dụng để thêm các item của dictionary 2 vào Dictionary đầu tiên
10	<u>Phương thức dict.values()</u> Trả về tất cả các value của một Dictionary

Date & Time trong Python

Với Python, bạn có thể dễ dàng thu được Date và Time hiện tại. Chương này sẽ giới thiệu một số phương thức được phổ biến trong khi làm việc với Date và Time bởi sử dụng Python.

Lấy Time hiện tại trong Python

Để lấy Time hiện tại, bạn sử dụng hàm tiền định nghĩa localtime(). Hàm localtime() này nhận một tham số là time.time(). Ở đây, time là module, time() là một hàm mà trả về system time hiện tại được biểu diễn dưới dạng số tick (số tích tắc) từ 12:00 am, 1/1/1970. Về cơ bản, tick là một số thực.

```
import time;
localtime = time.localtime(time.time())
print "Thoi gian hien tai la :", localtime
```

Kết quả là:

```
Thoi gian hien tai la : time.struct_time(tm_year=2015, tm_mon=11, tm_mday=29, tm_hour=19,
tm_min=16, tm_sec=54, tm_wday=6, tm_yday=333, tm_isdst=0)
```

Dưới đây là phần giải thích:

Time được trả về là một cấu trúc gồm 9 thuộc tính. Như trong bảng sau:

Thuộc tính	Miêu tả
tm_year	Trả về năm hiện tại (ví dụ: 2015)
tm_mon	Trả về tháng hiện tại (1-12)
tm_mday	Trả về ngày hiện tại (1-31)
tm_hour	Trả về giờ hiện tại (0-23)

tm_min	Trả về phút hiện tại (0-59)
tm_sec	Trả về giây hiện tại (0-61 với 60 và 61 là các dây nhuận)
tm_wday	Trả về ngày trong tuần (0-6 với 0 là Monday)
tm_yday	Trả về ngày trong năm (1-366 kể cả năm nhuận)
tm_isdst	Trả về -1, 0 hoặc 1 tức là có xác định DST không

Lấy Time đã được định dạng trong Python

Bạn có thể định dạng bất kỳ time nào theo yêu cầu của bạn, nhưng phương thức đơn giản nhất là `asctime()`. Đây là một hàm đã được định nghĩa trong `time module`. Hàm này trả về một time đã được định dạng bao gồm ngày trong tuần, tháng, ngày trong tháng, thời gian và năm. Ví dụ:

```
import time;

localtime = time.asctime( time.localtime(time.time()) )
print "Thoi gian da duoc dinh dang la :", localtime
```

Kết quả là:

```
Thoi gian da duoc dinh dang la : Sun Nov 29 19:16:30 2015
```

time module trong Python

Có nhiều hàm được định nghĩa sẵn trong `time` Module mà bạn có thể được sử dụng để làm việc với time.

STT	Hàm và Miêu tả
1	<u>Hàm time.altzone</u> Trả về offset của DST timezone (số giây)

2	<u>Hàm time.asctime([tupletime])</u> Chấp nhận một time-tuple và trả về một chuỗi gồm 24 ký tự có thể đọc được ví dụ như Mon Dec 11 18:07:14 2015
3	<u>Hàm time.clock()</u> Trả về CPU time hiện tại dưới dạng số giây dạng số thực
4	<u>Hàm time.ctime([secs])</u> Giống asctime(localtime(secs)) và nếu không có tham số thì giống như asctime()
5	<u>Hàm time.gmtime([secs])</u> Chuyển đổi một time được biểu diễn là số giây từ epoch sang một struct_time trong UTC
6	<u>Hàm time.localtime([secs])</u> Tương tự như gmtime(), nhưng nó chuyển đổi số giây thành local time.
7	<u>Hàm time.mktime(tupletime)</u> Là ngược với hàm localtime(). Trả về một số thực để tương thích với time()
8	<u>Hàm time.sleep(secs)</u> Dừng trình thực thi trong số giây đã cho là secs
9	<u>Hàm time.strftime(fmt[,tupletime])</u> Chuyển đổi một tuple hoặc struct_time thành một chuỗi được xác định bởi tham số format

10	<u>Hàm time.strptime(str,fmt='%a %b %d %H:%M:%S %Y')</u> Parse một chuỗi biểu diễn time theo một định dạng đã cho
11	<u>Hàm time.time()</u> Trả về time dưới dạng một số thực được diễn đạt bởi số giây từ epoch, trong UTC
12	<u>Hàm time.tzset()</u> Phục hồi các qui ước về thời gian được sử dụng bởi các chương trình con của thư viện. Biến môi trường TZ xác định cách được thực hiện

Có hai thuộc tính quan trọng có sẵn với time Module là:

- **time.timezone**: Thuộc tính time.timezone là số giây trong local timezone (không DST) từ UTC (>0 trong Americas; <=0 trong Europe, Asia, Africa).
- **time.tzname**: Thuộc tính time.tzname là một cặp các chuỗi biểu diễn locale và biểu diễn phụ thuộc, mà tương ứng là tên của local timezone với và không với DST.

calendar Module trong Python

Python cung cấp calendar Module để giúp bạn hiển thị Calendar. Ví dụ:

```
import calendar  
  
print "Thang hien tai la:"  
  
cal = calendar.month(2014, 6)  
  
print cal
```

Kết quả là:

```
Thang hien tai la:  
November 2015  
Mo Tu We Th Fr Sa Su  
1  
2 3 4 5 6 7 8
```

```
9 10 11 12 13 14 15  
16 17 18 19 20 21 22  
23 24 25 26 27 28 29  
30
```

Có rất nhiều hàm và phương thức đã được xây dựng sẵn trong calendar Module giúp bạn làm việc với Calendar. Dưới đây là một số hàm và phương thức:

Hàm	Miêu tả
Hàm prcal(year)	In cả calendar của năm
Hàm firstweekday()	Trả về ngày trong tuần đầu tiên. Theo mặc định là 0 mà xác định là Monday
Hàm isleap(year)	Trả về true nếu năm đã cho là năm nhuận, nếu không là false
Hàm monthcalendar(year,month)	Trả về một list gồm các ngày trong tháng đã cho của năm dưới dạng các tuần
Hàm leapdays(year1,year2)	Trả về số ngày nhuận từ năm year1 tới năm year2
Hàm prmonth(year,month)	In ra tháng đã cho của năm đã cung cấp

Dưới đây là ví dụ cho một số hàm:

Hàm firstweekday()

```
import calendar  
print calendar.firstweekday()
```

Kết quả là:

```
0
```

Hàm isleap(year)

```
import calendar  
print calendar.isleap(2000)
```

Kết quả là:

```
True
```

Hàm monthcalendar(year,month)

```
import calendar  
print calendar.monthcalendar(2015,11)
```

Kết quả là:

```
[[0, 0, 0, 0, 0, 0, 1], [2, 3, 4, 5, 6, 7, 8], [9, 10, 11, 12, 13, 14, 15], [16, 17, 18, 19, 20, 21, 22], [23, 24, 25, 26, 27, 28, 29], [30, 0, 0, 0, 0, 0, 0]]
```

Hàm prmonth(year,month)

```
import calendar  
print calendar.prmonth(2015,11)
```

Kết quả là:

```
November 2015  
Mo Tu We Th Fr Sa Su  
1  
2 3 4 5 6 7 8  
9 10 11 12 13 14 15  
16 17 18 19 20 21 22  
23 24 25 26 27 28 29  
30  
None
```

Hàm trong Python

Hàm, là một khối code được tổ chức và có thể tái sử dụng, để thực hiện một hành động nào đó. Trong các chương trước, bạn đã làm quen với một số hàm đã được xây dựng sẵn trong Python, điển hình như hàm print(). Tuy nhiên bạn cũng có thể tạo riêng cho mình một hàm với cách định nghĩa và kiểu giá trị cho riêng bạn. Các hàm này được gọi là user-defined function.

Định nghĩa một hàm trong Python

Khi định nghĩa các hàm để cung cấp một tính năng nào đó, bạn cần theo các qui tắc sau:

- Từ khóa **def** được sử dụng để bắt đầu phần định nghĩa hàm. Def xác định phần bắt đầu của khối hàm.
- def được theo sau bởi **ten_ham** được theo sau bởi các dấu ngoặc đơn () .
- Các tham số được truyền vào bên trong các dấu ngoặc đơn. Ở cuối là dấu hai chấm.
- Trước khi viết một code, một độ thut dòng được cung cấp trước mỗi lệnh. Độ thut dòng này nên giống nhau cho tất cả các lệnh bên trong hàm đó.
- Lệnh đầu tiên của hàm là tùy ý, và nó là Documentation String của một hàm đó.
- Sau đó là lệnh để được thực thi.

Cú pháp

```
def ten_ham( cac_tham_so ):  
    "function_docstring"  
    function_suite  
    return [bieu_thuc]
```

Ví dụ

Hàm sau nhận một chuỗi là tham số input và in nó trên màn hình chuẩn:

```
def printme( str ):  
    "Chuoi nay duoc truyen vao trong ham"  
    print str  
    return
```

Triệu hồi một hàm trong Python

Để thực thi một hàm, bạn cần gọi hàm đó. Phân định nghĩa hàm cung cấp thông tin về tên hàm các tham số và định nghĩa những hoạt động nào được thực hiện bởi hàm đó. Để thực thi phân định nghĩa của hàm, bạn cần gọi hàm đó. Cú pháp như sau:

```
ten_ham( cac_tham_so )
```

Ví dụ sau minh họa cách gọi hàm printme():

```
# Phan dinh nghia ham o day

def printme( str ):
    "Chuoi nay duoc truyen vao trong ham"
    print str
    return;

# Bay gio ban co the goi ham printme
printme("Loi goi dau tien toi custom func!")
printme("Loi goi thu hai toi custom func")
```

Khi code trên được thực thi sẽ cho kết quả:

```
Loi goi dau tien toi custom func!
Loi goi thu hai toi custom func
```

Hàm return() trong Python

Hàm return(bieu_thuc) được sử dụng để gửi điều khiển quay trở lại người gọi với bieu_thuc đã cho. Trong trường hợp không cung cấp bieu_thuc, thì hàm return này sẽ trả về None. Nói cách khác, lệnh return được sử dụng để thoát khỏi định nghĩa hàm.

Các ví dụ trên không trả về bất cứ giá trị nào. Bạn có thể trả về một giá trị từ một hàm như sau:

```
# Phan dinh nghia ham o day
```

```
def sum( arg1, arg2 ):  
    # Cong hai tham so va tra ve ket qua."  
    total = arg1 + arg2  
    print "Ben trong ham : ", total  
    return total;  
  
# Bay gio ban co the goi ham sum nay  
total = sum( 10, 20 );  
print "Ben ngoai ham : ", total
```

Kết quả là:

```
Ben trong ham : 30  
Ben ngoai ham : 30
```

Phân biệt argument và parameter trong Python

Có hai kiểu dữ liệu được truyền trong hàm:

- Kiểu dữ liệu đầu tiên là dữ liệu được truyền trong lời gọi hàm. Dữ liệu này được gọi là argument. Argument có thể là hằng, biến hoặc biểu thức.
- Kiểu dữ liệu thứ hai là dữ liệu được nhận trong phần định nghĩa hàm. Dữ liệu này được gọi là parameter. Parameter phải là biến để giữ các giá trị đang đến.

Ví dụ:

```
def addition(x,y):  
    print x+y  
x=15  
addition(x ,10)  
addition(x,x)  
y=20  
addition(x,y)
```

Kết quả là:

```
>>>
```

```
25  
30  
35  
>>>
```

Với cách phân biệt trên, bạn có thể hiểu hơn về đâu là kiểu tham số bạn đang dùng, còn thực sự thì mình nghĩ nó cũng không quan trọng lắm. Do đó, trong phần tiếp theo, chúng tôi sẽ gọi chung chúng là tham số.

Truyền bởi tham chiếu vs bởi giá trị trong Python

Tất cả parameter (argument) trong Python được truyền bởi tham chiếu. Nghĩa là nếu bạn thay đổi những gì mà một parameter tham chiếu tới bên trong một hàm, thì thay đổi này cũng phản ánh trong hàm đang gọi. Ví dụ:

```
# Phan dinh nghia ham o day  
  
def changeme( mylist ):  
    "Thay doi list da truyen cho ham nay"  
    mylist.append([1,2,3,4]);  
    print "Cac gia tri ben trong ham la: ", mylist  
    return  
  
  
# Bay gio ban co the goi ham changeme function  
mylist = [10,20,30];  
changeme( mylist );  
print "Cac gia tri ben ngoai ham la: ", mylist
```

Ở đây, chúng ta đang duy trì tham chiếu của đối tượng đang truyền và đang phụ thêm các giá trị trong cùng đối tượng đó. Vì thế đoạn code này sẽ cho kết quả:

```
Cac gia tri ben trong ham la: [10, 20, 30, [1, 2, 3, 4]]  
Cac gia tri ben ngoai ham la: [10, 20, 30, [1, 2, 3, 4]]
```

Bạn theo dõi một ví dụ nữa, tại đây tham số đang được truyền bởi tham chiếu và tham chiếu đang được ghi đè bên trong hàm được gọi.

```
# Phan dinh nghia ham o day
def changeme( mylist ):
    "Thay doi list da truyen cho ham nay"
    mylist = [1,2,3,4]; # Lenh nay gan mot tham chieu moi cho mylist
    print "Cac gia tri ben trong ham la: ", mylist
    return

# Bay gio ban co the goi ham changeme
mylist = [10,20,30];
changeme( mylist );
print "Cac gia tri ben ngoai ham la: ", mylist
```

Tham số mylist là cục bộ (local) với hàm changeme. Việc thay đổi mylist bên trong hàm này không ảnh hưởng tới mylist. Và cuối cùng code trên sẽ cho kết quả:

```
Cac gia tri ben trong ham la: [1, 2, 3, 4]
Cac gia tri ben ngoai ham la: [10, 20, 30]
```

Phạm vi biến trong Python

Tất cả các biến trong một chương trình không phải là có thể truy cập tại tất cả vị trí ở trong chương trình đó. Điều này phụ thuộc vào nơi bạn đã khai báo một biến.

Phạm vi biến quyết định nơi nào của chương trình bạn có thể truy cập một định danh cụ thể. Trong Python, có hai khái niệm về phạm vi biến là:

- Biến toàn cục
- Biến cục bộ

Biến cục bộ trong Python

Các biến được khai báo bên trong một thân hàm là biến cục bộ. Tức là các biến cục bộ này chỉ có thể được truy cập ở bên trong hàm mà bạn đã khai báo, không thể được truy cập ở bên ngoài thân hàm đó. Ví dụ:

```
def msg():
```

```
a=10  
print "Gia tri cua a la",a  
return  
  
msg()  
print a #no se cho mot error vi bien la cuc bo
```

Kết quả sẽ cho một lỗi vì biến là cục bộ.

```
>>>  
Gia tri cua a la 10  
  
Traceback (most recent call last):  
  File "C:/Python27/lam.py", line 7, in >module<  
    print a #no se cho mot error vi bien la cuc bo  
TenError: name 'a' is not defined  
>>>
```

Biến toàn cục trong Python

Biến được định nghĩa bên ngoài hàm được gọi là biến toàn cục. Biến toàn cục có thể được truy cập bởi tất cả các hàm ở khắp nơi trong chương trình. Do đó phạm vi của biến toàn cục là rộng nhất. Ví dụ:

```
b=20  
def msg():  
    a=10  
    print "Gia tri cua a la",a  
    print "Gia tri cua b la",b  
    return  
  
msg()  
print b
```

Kết quả là:

```
>>>
```

```
Gia tri cua a la 10
Gia tri cua b la 20
20
>>>
```

Tham số hàm trong Python

Sau khi đã tìm hiểu về phạm vi biến, tiếp theo chúng ta cùng tìm hiểu về các loại tham số hàm. Python hỗ trợ các kiểu tham số chính thức sau:

- Tham số bắt buộc
- Tham số mặc định
- Tham số từ khóa (tham số được đặt tên)
- Số tham số thay đổi

Tham số bắt buộc trong Python

Các tham số bắt buộc là các tham số được truyền tới một hàm theo một thứ tự chính xác. Ở đây, số tham số trong lời gọi hàm nên kết nối chính xác với phần định nghĩa hàm.

Bạn theo dõi ví dụ dưới đây:

```
#Phan dinh nghia cua ham sum
def sum(a,b):
    "Ham co hai tham so"
    c=a+b
    print c

sum(10,20)
sum(20)
```

Khi code trên được thực thi sẽ cho một lỗi như sau:

```
>>>
30
```

```
Traceback (most recent call last):
  File "C:/Python27/su.py", line 8, in >module<
    sum(20)
TypeError: sum() takes exactly 2 arguments (1 given)
>>>
```

Giải thích:

1, Trong trường hợp đầu tiên, khi hàm sum() được gọi đã được truyền hai giá trị là 10 và 20, đầu tiên Python so khớp với phần định nghĩa hàm, sau đó 10 và 20 được gán tương ứng cho a và b. Do đó hàm sum được tính toán và được in.

2, Trong trường hợp thứ hai, khi bạn chỉ truyền cho hàm sum() một giá trị là 20, giá trị này được truyền tới phần định nghĩa hàm. Tuy nhiên phần định nghĩa hàm chấp nhận hai tham số trong khi chỉ có một giá trị được truyền, do đó sẽ tạo ra một lỗi như trên.

Tham số mặc định trong Python

Tham số mặc định là tham số mà cung cấp các giá trị mặc định cho các tham số được truyền trong phần định nghĩa hàm, trong trường hợp mà giá trị không được cung cấp trong lời gọi hàm. Ví dụ:

```
#Phan dinh nghia ham
def msg(Id,Ten,Age=21):
    "In gia tri da truyen"
    print Id
    print Ten
    print Tuoi
    return

#Function call
msg(Id=100,Ten='Hoang',Tuoi=20)
msg(Id=101,Ten='Thanh')
```

Kết quả là:

```
>>>
100
Hoang
```

```
20  
101  
Thanh  
21  
>>>
```

Giải thích:

1, Trong trường hợp đầu tiên, khi hàm msg() được gọi đang truyền ba giá trị là 100, Hoang, và 20, thì các giá trị này sẽ được gán tương ứng cho các tham số và do đó chúng được in ra tương ứng.

2, Trong trường hợp thứ hai, khi bạn chỉ truyền hai tham số cho hàm msg() được gọi là 101 và Thanh, thì các giá trị này được gán tương ứng cho ID và Ten. Không có giá trị nào được gán cho tham số thứ ba trong lời gọi hàm, và vì thế hàm sẽ lấy giá trị mặc định là 21.

Tham số từ khóa trong Python

Sử dụng tham số từ khóa, tham số được truyền trong lời gọi hàm được kết nối với phần định nghĩa hàm dựa trên **tên của tham số**. Với trường hợp này, vị trí của các tham số trong lời gọi hàm là tùy ý. Ví dụ:

```
def msg(id,name):  
    "In gia tri da truyen"  
    print id  
    print ten  
    return  
  
msg(id=100,ten='Hoang')  
msg(ten='Thanh',id=101)
```

Kết quả là:

```
>>>  
100  
Hoang  
101  
Thanh  
>>>
```

Giải thích:

1, Trong trường hợp đầu tiên, trong lời gọi hàm msg(), bạn đã truyền hai giá trị và truyền vị trí giống như của chúng trong phần định nghĩa hàm. Sau khi so khớp với phần định nghĩa hàm, thì các giá trị này được truyền tương ứng với các tham số trong phần định nghĩa hàm. Điều này được thực hiện dựa trên tên tham số.

2, Trong trường hợp thứ hai, trong lời gọi hàm msg(), bạn cũng cung cấp hai giá trị nhưng với vị trí khác với phần định nghĩa hàm. Thì ở đây, dựa vào tên của tham số, các giá trị này cũng được truyền tương ứng cho các tham số trong phần định nghĩa hàm.

Hàm với số tham số thay đổi trong Python

Bạn có thể cần xử lý một hàm mà có số tham số nhiều hơn là bạn đã xác định trong khi định nghĩa hàm. Những tham số này được gọi là các tham số có số tham số thay đổi (variable-length args) và không được đặt tên trong định nghĩa hàm, không giống như các tham số bắt buộc và tham số mặc định.

Cú pháp cho một hàm có số thay đổi là:

```
def tenham([tham_so_chinh_thuc,] *var_args_tuple ):  
    "function_docstring"  
    function_suite  
    return [bieu_thuc]
```

Một dấu * được đặt trước tên biến để giữ các giá trị của các tham số loại này. Tuple này vẫn là **trống** nếu không có tham số bổ sung nào được xác định trong khi gọi hàm. Dưới đây là ví dụ đơn giản.

```
# Phan dinh nghia ham o day  
  
def printinfo( arg1, *vartuple ):  
    "In mot tham so da truyen"  
    print "Ket qua la: "  
    print arg1  
    for var in vartuple:  
        print var
```

```
return;

# Bay gio ban co the goi ham printinfo
printinfo( 10 )
printinfo( 70, 60, 50 )
```

Kết quả là:

```
Ket qua la:
10
Ket qua la:
70
60
50
```

Hàm nặc danh trong Python

Hàm nặc danh (hàm vô danh), hiểu theo cách đơn giản, là hàm không có tên và chúng không được khai báo theo cách chính thức bởi từ khóa **def**. Để khai báo hàm này, bạn sử dụng từ khóa **lambda**. Lambda nhận bất kỳ lượng tham số nào và chỉ trả về một giá trị trong dạng một biểu thức đã được ước lượng. Bạn không thể gọi trực tiếp gọi hàm nặc danh để in bởi vì lambda cần một biểu thức. Ngoài ra, các hàm lambda có namespace cục bộ của chúng. Dưới đây là cú pháp của hàm lambda:

```
lambda [arg1 [,arg2,.....argn]]:bieu_thuc
```

Bạn theo dõi ví dụ sau để hiểu cách hàm lambda làm việc:

```
#Phan dinh nghia ham
square=lambda x1: x1*x1

#Goi square nhu la mot ham
print "Binh phuong cua so la",square(10)
```

Kết quả là:

```
>>>
Binh phuong cua so la 100
```

>>>

So sánh hàm chính thức và hàm nặc danh trong Python?

Bạn theo dõi hai ví dụ sau:

Ví dụ cho hàm chính thức:

```
#Phan dinh nghia ham

def square(x):
    return x*x

#Goi ham square
print "Binh phuong cua so la",square(10)
```

Ví dụ cho hàm nặc danh:

```
#Phan dinh nghia ham
square=lambda x1: x1*x1

#Goi square nhu la mot ham
print "Binh phuong cua so la",square(10)
```

Giải thích:

Hàm nặc danh được tạo bởi sử dụng từ khóa lambda, không phải bởi từ khóa def. Hàm này chỉ trả về một giá trị dưới dạng một biểu thức đã được ước lượng.

Module trong Python

Module được sử dụng để phân loại code thành các phần nhỏ hơn liên quan với nhau. Hay nói cách khác, Module giúp bạn tổ chức Python code một cách logic để giúp bạn dễ dàng hiểu và sử dụng code đó hơn. Trong Python, Module là một đối tượng với các thuộc tính mà bạn có thể đặt tên tùy ý và bạn có thể gán kết và tham chiếu.

Về cơ bản, một Module là một file, trong đó các lớp, hàm và biến được định nghĩa. Tất nhiên, một Module cũng có thể bao gồm code có thể chạy.

Bạn theo dõi qua ví dụ sau: Nếu nội dung của một quyển sách không được lập chỉ mục hoặc phân loại thành các chương riêng, thì quyển sách này có thể trở nên nhảm chán và gây khó khăn cho độc giả khi đọc và hiểu nó. Tương tự, Module trong Python là các file mà có các code tương tự nhau, hay có liên quan với nhau. Chúng có lợi thế sau:

- **Khả năng tái sử dụng:** Module có thể được sử dụng ở trong phần Python code khác, do đó làm tăng tính tái sử dụng code.
- **Khả năng phân loại:** Các kiểu thuộc tính tương tự nhau có thể được đặt trong một Module.

Ví dụ

Tiếp theo, bạn theo dõi một ví dụ về một Module có tên là vietjack được đặt bên trong test.py.

```
def print_func( par ):  
    print "Hello : ", par  
    return
```

Để import một Module, bạn có thể sử dụng một trong ba cách dưới đây:

Sử dụng lệnh import trong Python

Bạn có thể sử dụng bất cứ source file nào dưới dạng như một Module bằng việc thực thi một lệnh import trong source file khác. Cú pháp của lệnh import là:

```
import module1[, module2,... moduleN]
```

Giả sử mình có đoạn code sau:

```
def add(a,b):
```

```
c=a+b  
print c  
return
```

Lưu file dưới tên là addition.py. Lệnh import được sử dụng như sau với file này:

```
import addition  
addition.add(10,20)  
addition.add(30,40)
```

Ở đây, trong addition.add() thì addition là tên file và add() là phương thức đã được định nghĩa trong addition.py. Do đó, bạn có thể sử dụng phương thức đã được định nghĩa trong Module bằng cách là ten_file.phuong_thuc(). Code trên sẽ cho kết quả:

```
>>>  
30  
70  
>>>
```

Ghi chú: Bạn có thể truy cập bất cứ hàm nào bên trong một Module theo phương thức như trên.

Để import nhiều Module, bạn sử dụng cách như trong ví dụ sau:

1, msg.py

```
def msg_method():  
    print "Hom nay troi mua"  
    return
```

2, display.py

```
def display_method():  
    print "Thoi tiet kha am uot"  
    return
```

3, multiimport.py

```
import msg,display  
msg.msg_method()
```

```
display.display_method()
```

Kết quả là:

```
>>>  
Hom nay troi mua  
Thoi tiet kha am uot  
>>>
```

Sử dụng lệnh from...import trong Python

Lệnh **from...import** được sử dụng để import thuộc tính cụ thể từ một Module. Trong trường hợp mà bạn không muốn import toàn bộ Module nào đó thì bạn có thể sử dụng lệnh này. Cú pháp của lệnh **from...import** là:

```
from modname import name1[, name2[, ... nameN]]
```

Dưới đây là ví dụ:

1, area.py

```
def circle(r):  
    print 3.14*r*r  
    return  
  
def square(l):  
    print l*l  
    return  
  
def rectangle(l,b):  
    print l*b  
    return  
  
def triangle(b,h):  
    print 0.5*b*h  
    return
```

2, area1.py

```
from area import square,rectangle  
square(10)  
rectangle(2,5)
```

Kết quả là:

```
>>>  
100  
10  
>>>
```

Sử dụng lệnh from...import* trong Python

Sử dụng lệnh này, bạn có thể import toàn bộ Module. Do đó bạn có thể truy cập các thuộc tính trong Module này. Cú pháp của lệnh là:

```
from modname import *
```

Ví dụ dưới đây chúng ta sẽ import **area.py** ở trên:

2, area1.py

```
from area import *  
square(10)  
rectangle(2,5)  
circle(5)  
triangle(10,20)
```

Kết quả là:

```
>>>  
100  
10  
78.5  
100.0  
>>>
```

Built-in Module trong Python

Phần trên, bạn đã tìm hiểu cách tạo ra Module cho riêng mình và cách import chúng. Phần này sẽ giới thiệu các Module đã được xây dựng sẵn trong Python. Đó là math, random, threading, collections, os, mailbox, string, time, ... Mỗi Module này đã định nghĩa sẵn rất nhiều hàm để bạn có thể sử dụng để thực hiện các tính năng khác nhau. Bạn theo dõi một số ví dụ với hai Module là math và random mà có các hàm đã được giới thiệu trong các chương trước.

Ví dụ sử dụng math Module:

```
import math

a=4.6

print math.ceil(a)
print math.floor(a)

b=9

print math.sqrt(b)
print math.exp(3.0)
print math.log(2.0)
print math.pow(2.0,3.0)
print math.sin(0)
print math.cos(0)
print math.tan(45)
```

Ví dụ sử dụng random Module:

```
import random

print random.random()
print random.randint(2,8)
```

Package trong Python

Về cơ bản, một Package là một tập hợp các Module, sub-package, ... tương tự nhau. Đó là một cấu trúc có thứ bậc của thư mục và file.

Dưới đây là các bước để tạo và import một Package:

Bước 1: Tạo một thư mục, có tên là vietjack chẳng hạn.

Bước 2: Đặt các module khác nhau bên trong thư mục vietjack này. Chúng ta đặt ba Module là msg1.py, msg2.py, và msg3.py và đặt tương ứng code trên vào các Module tương ứng. Bạn đặt hàm msg1() trong msg1.py, hàm msg2() trong msg2.py và hàm msg3() trong msg3.py.

Bước 3: Tạo một __init__.py file để xác định các thuộc tính trong mỗi Module.

Bước 4: Cuối cùng bạn import package này và sử dụng các thuộc tính đó bởi sử dụng package.

Bạn theo dõi ví dụ sau:

1, Tạo thư mục

```
import os  
os.mkdir("Info")
```

2, Đặt các module khác nhau trong package:

msg1.py

```
def msg1():  
    print "Day la msg1"
```

msg2.py

```
def msg2():  
    print "Day la msg2"
```

msg3.py

```
def msg3():  
    print "Day la msg3"
```

3, Tạo một __init__.py file.

```
from msg1 import msg1  
from msg2 import msg2  
from msg3 import msg3
```

4, Import package này và sử dụng các thuộc tính.

```
import Info
```

```
Info.msg1()  
Info.msg2()  
Info.msg3()
```

Kết quả là:

```
>>>  
Day la msg1  
Day la msg2  
Day la msg3  
>>>
```

Câu hỏi: __init__.py file là gì?

Nó đơn giản là một file được sử dụng để xem xét các thư mục trên disk dưới dạng package của Python. Về cơ bản, file này được sử dụng để khởi tạo các Package trong Python.

File I/O trong Python

Chắc bạn cũng đã quen thuộc với khái niệm File I/O khi đã học qua C hoặc C++. Python cũng hỗ trợ việc đọc và ghi dữ liệu tới các file.

In kết quả ra màn hình trong Python

Đến đây, chắc bạn đã quá quen thuộc về cách sử dụng của lệnh print. Lệnh này được sử dụng để in kết quả trên màn hình. Hàm này chuyển đổi biểu thức mà bạn đã truyền cho nó thành dạng chuỗi và ghi kết quả trên đầu ra chuẩn Standard Output. Cú pháp của lệnh print là:

```
print "Hoc Python la kha don gian," , "ban co thay vay khong?"
```

Kết quả là:

```
Hoc Python la kha don gian, ban co thay vay khong?
```

Đọc input từ bàn phím trong Python

Python cung cấp hai hàm đã được xây dựng sẵn để nhận input từ người dùng. Hai hàm đó là:

- Hàm input()
- Hàm raw_input()

Hàm input() trong Python

Hàm này được sử dụng để nhận input từ người dùng. Hàm này giống hàm raw_input(), nhưng với hàm input() này thì bất cứ biểu thức nào được nhập từ người dùng thì nó ước lượng và sau đó trả về kết quả. Ví dụ:

```
str = input("Nhap dau vao cua ban: ");
print "Dau vao da nhan la : ", str
```

Code trên sẽ cho kết quả sau tùy thuộc vào input bạn đã nhập:

```
Nhap dau vao cua ban: [x*5 for x in range(2,10,2)]
```

Dau vao da nhan la : [10, 20, 30, 40]

Hàm raw_input() trong Python

Hàm raw_input() được sử dụng để nhận đầu vào từ người dùng. Nó nhận đầu vào từ Standard Input dưới dạng một chuỗi và đọc dữ liệu từ từng dòng một. Ví dụ:

```
str = raw_input("Nhap dau vao cua ban: ");
print "Dau vao da nhan la : ", str
```

Kết quả khi mình nhập "Hello Python!" là:

```
Nhap dau vao cua ban: Hello Python
Dau vao da nhan la : Hello Python
```

Ghi chú: Hàm raw_input() trả về một chuỗi. Vì thế trong trường hợp một biểu thức cần được ước lượng, thì nó phải ép kiểu sang kiểu dữ liệu sau của nó. Bạn theo dõi một số ví dụ dưới đây.

Ví dụ 1:

```
prn=int(raw_input("Trang VietJack"))
r=int(raw_input("Thu Tu"))
t=int(raw_input("Vi Tri"))
si=(prn*r*t)/100
print "VietJack Chao Ban ",si
```

Kết quả là:

```
>>>
Trang VietJack1000
Thu Tu10
Vi Tri2
VietJack Chao Ban  200
>>>
```

Ví dụ 2:

```
name=raw_input("Nhập tên bạn ")  
math=float(raw_input("Nhập điểm môn Toán"))  
physics=float(raw_input("Nhập điểm môn Vật Lý"))  
chemistry=float(raw_input("Nhập điểm môn Hóa Học"))  
rollno=int(raw_input("Nhập mssv"))  
  
print "Welcome ",name  
print "MSSV của bạn là ",rollno  
print "Điểm môn Toán là ",math  
print "Điểm môn Vật Lý là ",physics  
print "Điểm môn Hóa Học là ",chemistry  
print "Điểm trung bình là ",(math+physics+chemistry)/3
```

Kết quả là:

```
>>>  
Nhập tên bạn Hoang  
Nhập điểm môn Toán7.68  
Nhập điểm môn Vật Lý7.14  
Nhập điểm môn Hóa Học8.84  
Nhập mssv0987645672  
Welcome Hoang  
MSSV của bạn là 987645672  
Điểm môn Toán là 7.68  
Điểm môn Vật Lý là 7.14  
Điểm môn Hóa Học là 8.84  
Điểm trung bình là 7.88666666667  
>>>
```

Làm việc với File trong Python

Python cung cấp nhiều cách tiện lợi để bạn làm việc với file. Ở trên, bạn đã đọc dữ liệu từ Standard Input và ghi dữ liệu tới Standard Output. Bây giờ chúng ta tìm hiểu cách sử dụng các file dữ liệu thực sự. Một file là một nơi lưu trữ ngoại vi trên hard disk, tại đó dữ liệu có thể được lưu trữ và thu nhận. Dưới đây là các hoạt động trên File:

Mở file trong Python

Trước khi làm việc với bất cứ File nào, bạn phải mở File đó. Để mở một File, Python cung cấp hàm `open()`. Nó trả về một đối tượng File mà được sử dụng với các hàm khác. Với File đã mở, bạn có thể thực hiện các hoạt động đọc, ghi, ... trên File đó. Cú pháp của hàm `open()` là:

```
doi_tuong_file = open(ten_file [, access_mode][, buffer])
```

Ở đây,

- **ten_file** là tên File bạn muốn truy cập.
- **access_mode** xác định chế độ của File đã được mở. Có nhiều mode sẽ được trình bày trong phần dưới. Bạn nên xác định mode này phụ thuộc vào các hoạt động mà bạn muốn thực hiện trên File đó. Chế độ truy cập mặc định là `read`.
- **buffer** Nếu buffer được thiết lập là 0, nghĩa là sẽ không có trình đệm nào diễn ra. Nếu xác định là 1, thì trình đệm dòng được thực hiện trong khi truy cập một File. Nếu là số nguyên lớn hơn 1, thì hoạt động đệm được thực hiện với kích cỡ bộ đệm đã cho. Nếu là số âm, thì kích cỡ bộ đệm sẽ là mặc định (hành vi mặc định).

Đóng một File trong Python

Khi bạn đã thực hiện xong các hoạt động trên File thì cuối cùng bạn cần đóng File đó. Python tự động đóng một File khi đối tượng tham chiếu của một File đã được tái gán cho một file khác. Tuy nhiên, sử dụng phương thức `close()` để đóng một file là một sự thực hành tốt cho bạn. Phương thức `close()` có cú pháp như sau:

```
fileObject.close();
```

Đọc một File trong Python

Để đọc một File, bạn sử dụng phương thức `read()` trong Python. Cú pháp là:

```
doi_tuong_file.read(giaTri);
```

Ở đây, `value` là số byte để được đọc từ file đã mở. Phương thức này bắt đầu đọc từ phần đầu file và nếu bạn không cung cấp tham số `value` thì phương thức này cố gắng đọc nhiều dữ liệu nhất có thể, có thể tới cuối File.

Ghi tới một File trong Python

Phương thức write() được sử dụng để ghi bất kỳ chuỗi nào tới một File đã mở. Bạn chú ý là phương thức write này không thêm một ký tự newline ('\n') vào cuối chuỗi. Cú pháp của write() là:

```
doi_tuong_file.write(string);
```

Dưới đây là chương trình ví dụ để đọc và ghi dữ liệu từ một File trong Python:

```
obj=open("abcd.txt","w")
obj.write("Python xin chao cac ban")
obj.close()
obj1=open("abcd.txt","r")
s=obj1.read()
print s
obj1.close()
obj2=open("abcd.txt","r")
s1=obj2.read(20)
print s1
obj2.close()
```

Kết quả là:

```
>>>
Python xin chao cac ban
Chao mung ban den voi the gioi Python
>>>
```

Các thuộc tính của File trong Python

Đối tượng File có các thuộc tính sau:

Thuộc tính	Miêu tả
file.closed	Trả về true nếu file đã được đóng, nếu không là false

file.mode	Trả về chế độ truy cập nào mà file đã mở với
file.name	Trả về tên file
file.softspace	Trả về false nếu space được yêu cầu tương ứng với print, nếu không là true

Ví dụ

Chúng ta tạo *foo.txt* có nội dung sau:

```
# Mo mot file
fo = open("foo.txt", "wb")
fo.write( "Python la mot ngon ngu lap trinh tuyet voi.\nMinh cung nghi nhu the!!\n");

# Dong file da mo
fo.close()
```

Giờ chúng ta kiểm tra các thuộc tính của nó:

```
# Mo mot file
fo = open("foo.txt", "wb")
print "Ten cua file la: ", fo.name
print "File da duoc dong hay chua : ", fo.closed
print "Che do mode la : ", fo.mode
print "Softspace la : ", fo.softspace
```

Kết quả là:

```
Ten cua file la: foo.txt
File da duoc dong hay chua : False
Che do mode la : wb
Softspace la : 0
```

Các chế độ truy cập (mode) của File trong Python

File có thể được mở với các chế độ truy cập khác nhau. File có thể được mở trong Text Mode hoặc Binary Mode. Bảng dưới liệt kê và giới thiệu các chế độ này:

Mode	Miêu tả
r	Mở file trong chế độ đọc, đây là chế độ mặc định. Con trỏ tại phần bắt đầu của File
rb	Mở file trong chế độ đọc cho định dạng nhị phân, đây là chế độ mặc định. Con trỏ tại phần bắt đầu của File
r+	Mở file để đọc và ghi. Con trỏ tại phần bắt đầu của File
rb+	Mở file để đọc và ghi trong định dạng nhị phân. Con trỏ tại phần bắt đầu của File
w	Mở File trong chế độ ghi. Nếu file đã tồn tại, thì ghi đè nội dung của file đó, nếu không thì tạo một file mới
wb	Mở File trong chế độ ghi trong định dạng nhị phân. Nếu file đã tồn tại, thì ghi đè nội dung của file đó, nếu không thì tạo một file mới
w+	Mở file để đọc và ghi. Nếu file tồn tại thì ghi đè nội dung của nó, nếu file không tồn tại thì tạo một file mới để đọc và ghi
wb+	Mở file để đọc và ghi trong định dạng nhị phân. Nếu file tồn tại thì ghi đè nội dung của nó, nếu file không tồn tại thì tạo một file mới để đọc và ghi
a	Mở file trong chế độ append. Con trỏ là ở cuối file nếu file này đã tồn tại. Nếu file không tồn tại, thì tạo một file mới để ghi
ab	Mở file trong chế độ append trong chế độ nhị phân. Con trỏ là ở cuối file nếu file này đã tồn tại. Nếu file không tồn tại, thì tạo một file mới để ghi

a+	Mở file trong để đọc và append. Con trỏ file tại cuối nếu file đã tồn tại. Nếu không tồn tại thì tạo một file mới để đọc và ghi
ab+	Mở file trong để đọc và append trong định dạng nhị phân. Con trỏ file tại cuối nếu file đã tồn tại. Nếu không tồn tại thì tạo một file mới để đọc và ghi

Thay tên file trong Python

Phương thức *rename()* trong os Module được sử dụng để thay tên file. Phương thức này nhận hai tham số là tên file cũ và tên file mới.

Cú pháp

```
os.rename(ten_file_hien_tai, ten_file_moi)
```

Ví dụ sau thay tên *test1.txt* thành *test2.txt*:

```
import os

# Thay tên từ test1.txt thành test2.txt
os.rename( "test1.txt", "test2.txt" )
```

Xóa file trong Python

Bạn có thể sử dụng phương thức *remove()* của os Module để xóa các file với tham số là tên file bạn cần xóa.

Cú pháp

```
os.remove(ten_file)
```

Ví dụ sau sẽ xóa *test2.txt*:

```
import os

# Xóa test2.txt
os.remove("text2.txt")
```

Vị trí File trong Python

Phương thức **tell()** nói cho bạn biết vị trí hiện tại bên trong file. Nói cách khác, việc đọc và ghi tiếp theo sẽ diễn ra trên các byte đó.

Phương thức **seek(offset[, from])** thay đổi vị trí hiện tại bên trong file. Tham số **offset** chỉ số byte để được di chuyển. Tham số **from** xác định vị trí tham chiếu mà từ đó byte được di chuyển.

Nếu **from** được thiết lập là 0 nghĩa là sử dụng phần đầu file như là vị trí tham chiếu và 1 nghĩa là sử dụng vị trí hiện tại như là vị trí tham chiếu và nếu là 2 thì sử dụng phần cuối file như là vị trí tham chiếu.

Ví dụ

Sử dụng **foo.txt** đã tạo ở trên để minh họa các hàm **tell** và **seek**:

```
# Mo mot file
fo = open("foo.txt", "r+")
str = fo.read(10);
print "Chuoi da doc la : ", str

# Kiem tra con tro hien tai
position = fo.tell();
print "Con tro file hien tai : ", position

# Dat lai vi tri con tro tai vi tri ban dau mot lan nua
position = fo.seek(0, 0);
str = fo.read(10);
print "Chuoi da doc la : ", str

# Dong file da mo
fo.close()
```

Kết quả là:

```
Chuoi da doc la : Python is
Con tro file hien tai : 10
```

Chuỗi da doc la : Python la

Thư mục trong Python

Tất cả file được chứa trong các thư mục đa dạng và Python cũng cung cấp rất nhiều phương thức để xử lý các hoạt động đa dạng liên quan tới thư mục. **os** Module có một số phương thức giúp bạn tạo, xóa, và thay đổi các thư mục.

Phương thức mkdir() trong Python

Bạn có thể sử dụng phương thức mkdir() của os Module để tạo các thư mục trong thư mục hiện tại. Bạn cần cung cấp một tham số là tên thư mục cho phương thức này.

Cú pháp

```
os.mkdir("thu_muc_moi")
```

Ví dụ sau tạo một thư mục *test* trong thư mục hiện tại.

```
import os

# Tao mot thu muc la "test"
os.mkdir("test")
```

Phương thức chdir() trong Python

Bạn có thể sử dụng phương thức chdir() để thay đổi thư mục hiện tại. Phương thức chdir() nhận một tham số là tên của thư mục bạn muốn tới từ thư mục hiện tại.

Cú pháp

```
os.chdir("thu_muc_moi")
```

Ví dụ sau tới thư mục /home/newdir.

```
import os

# Thay doi mot thu muc toi "/home/newdir"
os.chdir("/home/newdir")
```

Phương thức getcwd() trong Python

Phương thức getcwd() hiển thị thư mục đang làm việc hiện tại.

Cú pháp

```
os.getcwd()
```

Ví dụ sau hiển thị thư mục đang làm việc hiện tại.

```
import os

# Lenh nay se cung cap vi tri thu muc hien tai
os.getcwd()
```

Phương thức rmdir() trong Python

Phương thức rmdir() xóa thư mục mà có tên được truyền như là một tham số cho phương thức này.

Trước khi xóa thư mục, tất cả nội dung trong nó nên được xóa.

Cú pháp

```
os.rmdir('ten_thu_muc')
```

Ví dụ sau sẽ xóa thư mục /tmp/test. Bạn phải cung cấp tên đầy đủ của thư mục, nếu không phương thức này sẽ không tìm thấy thư mục đó và sẽ không có hoạt động xóa diễn ra.

```
import os

# Xoa thu muc "/tmp/test" .
os.rmdir( "/tmp/test" )
```

Các phương thức xử lý File và thư mục trong Python

Đối tượng File và OS cung cấp rất nhiều phương thức tiện ích để xử lý và thao tác với File và thư mục trên hệ điều hành Windows và Unix. Bạn truy cập đường link sau để tìm hiểu các phương thức này.

- Đối tượng File: cung cấp các phương thức để thao tác File.
- os Module: cung cấp rất nhiều phương thức để thao tác File và thư mục.

Standard Exception trong Python

Python cung cấp hai đặc điểm quan trọng là Xử lý ngoại lệ (Exception Handling) và Assertion để xử lý bất kỳ lỗi không mong đợi nào trong các chương trình Python của bạn và để thêm khả năng debug tới các chương trình đó.

Chương này chúng ta trước hết giới thiệu qua về các Exception chuẩn có trong Python. Bảng dưới đây liệt kê tất cả ngoại lệ chuẩn có sẵn trong Python:

Exception	Miêu tả
Exception	Lớp cơ sở (base class) của tất cả các ngoại lệ
StopIteration	Được tạo khi phương thức next() của một iterator không trả tới bất kỳ đối tượng nào
SystemExit	Được tạo bởi hàm sys.exit()
StandardError	Lớp cơ sở của tất cả exception có sẵn ngoại trừ StopIteration và SystemExit.
ArithmaticError	Lớp cơ sở của tất cả các lỗi xảy ra cho phép tính số học
OverflowError	Được tạo khi một phép tính vượt quá giới hạn tối đa cho một kiểu số
FloatingPointError	Được tạo khi một phép tính số thực thất bại
ZeroDivisionError	Được tạo khi thực hiện phép chia cho số 0 với tất cả kiểu số
AssertionError	Được tạo trong trường hợp lệnh assert thất bại
AttributeError	Được tạo trong trường hợp tham chiếu hoặc gán thuộc tính thất bại

EOFError	Được tạo khi không có input nào từ hàm raw_input() hoặc hàm input() và tới EOF (viết tắt của end of file)
ImportError	Được tạo khi một lệnh import thất bại
KeyboardInterrupt	Được tạo khi người dùng ngắt việc thực thi chương trình, thường là bởi nhấn Ctrl+c
LookupError	Lớp cơ sở cho tất cả các lỗi truy cứu
IndexError	Được tạo khi một chỉ mục không được tìm thấy trong một dãy (sequence)
KeyError	Được tạo khi key đã cho không được tìm thấy trong Dictionary
NameError	Được tạo khi một định danh không được tìm thấy trong local hoặc global namespace
UnboundLocalError	Được tạo khi cố gắng truy cập một biến cục bộ từ một hàm hoặc phương thức nhưng mà không có giá trị nào đã được gán cho nó
EnvironmentError	Lớp cơ sở cho tất cả ngoại lệ mà xuất hiện ở ngoài môi trường Python
IOError	Được tạo khi hoạt động i/o thất bại, chẳng hạn như lệnh print hoặc hàm open() khi cố gắng mở một file không tồn tại
OSError	Được do các lỗi liên quan tới hệ điều hành
SyntaxError	Được tạo khi có một lỗi liên quan tới cú pháp
IndentationError	Được tạo khi độ thụt dòng code không được xác định hợp lý
SystemError	Được tạo khi trình thông dịch tìm thấy một vấn đề nội tại, nhưng khi

	Lỗi này được bắt gặp thì trình thông dịch không thoát ra
SystemExit	Được tạo khi trình thông dịch thoát ra bởi sử dụng hàm sys.exit(). Nếu không được xử lý trong code, sẽ làm cho trình thông dịch thoát
TypeError	Được tạo khi một hoạt động hoặc hàm sử dụng một kiểu dữ liệu không hợp lệ
ValueError	Được tạo khi hàm đã được xây dựng sẵn có các kiểu tham số hợp lệ nhưng các giá trị được xác định cho tham số đó là không hợp lệ
RuntimeError	Được tạo khi một lỗi đã được tạo ra là không trong loại nào
NotImplementedError	Được tạo khi một phương thức abstract, mà cần được triển khai trong một lớp được kế thừa, đã không được triển khai thực sự

Assertion trong Python

Cách đơn giản nhất để nghĩ về một Assertion là xem nó giống như một lệnh raise-if-no. Một biểu thức được kiểm nghiệm, và nếu kết quả là false, thì một exception được tạo ra.

Một Assertion là một sanity-test mà bạn có thể bật hoặc tắt khi bạn được thực hiện với sự kiểm nghiệm chương trình của bạn.

Assertion được mang bởi lệnh assert. Các lập trình viên thường đặt các Assertion tại phần đầu của một hàm để kiểm tra tính hợp lệ của input, và sau một lời gọi hàm để kiểm tra tính hợp của output.

Lệnh assert trong Python

Khi bắt gặp lệnh assert, Python ước lượng biểu thức đi kèm, mà hy vọng rằng là true. Nếu biểu thức là false, thì Python tạo một ngoại lệ là **AssertionError**.

Dưới đây là cú pháp cho lệnh assert:

```
assert bieu_thuc[, cac_tham_so]
```

Nếu Assertion thất bại, Python sử dụng cac_tham_so là tham số cho AssertionError. Các ngoại lệ AssertionError có thể được bắt và được xử lý giống như bất kỳ ngoại lệ khác bởi sử dụng lệnh try-except, nhưng nếu không được xử lý thì chúng sẽ kết thúc chương trình và sẽ tạo một traceback.

Ví dụ

Dưới đây là một hàm để chuyển đổi từ độ K thành độ F.

```
def ChuyenKF(Nhietdo):
    assert (Nhietdo >= 0), "Lanh hon do khong tuyet doi!"
    return ((Nhietdo-273)*1.8)+32

print ChuyenKF(273)
print int(ChuyenKF(505.78))
print ChuyenKF(-5)
```

Khi code trên được thực thi sẽ cho kết quả:

```
32.0
451
Traceback (most recent call last):
  File "test.py", line 9, in <module>
    print ChuyenKF(-5)
  File "test.py", line 4, in ChuyenKF
    assert (Nhiетdo >= 0),"Lanh hon do khong tuyet doi!"
AssertionError: Lanh hon do khong tuyet doi!
```

Xử lý ngoại lệ (Exception Handling) trong Python

Exception là gì?

Ngoại lệ có thể là bất kỳ điều kiện bất thường nào trong chương trình mà phá vỡ luồng thực thi chương trình đó. Bất cứ khi nào một ngoại lệ xuất hiện, mà không được xử lý, thì chương trình ngừng thực thi và vì thế code không được thực thi.

Python đã định nghĩa sẵn rất nhiều ngoại lệ, mà đã được trình bày trong chương Standard Exception. Trong chương này chúng ta sẽ tìm hiểu cách xử lý ngoại lệ cũng như cách tạo các Custom Exception như thế nào.

Xử lý ngoại lệ trong Python

Nếu bạn thấy bất cứ code nào là khả nghi (có thể gây ra ngoại lệ) thì bạn có thể phòng thủ chương trình của mình bằng cách đặt các khối code khả nghi này trong một khối try. Khối try này được sau bởi lệnh except. Sau đó, nó được theo sau bởi các lệnh mà xử lý vấn đề đó.

Cú pháp

Dưới đây là cú pháp của khối try...except...else trong Python:

```
try:  
    Ban thuc hien cac hoat dong cua minh tai day;  
    Va day la phan code co the tao exception;  
    .....  
except ExceptionI:  
    Neu co ExceptionI, thi thuc thi khoi code nay  
except ExceptionII:  
    Neu co ExceptionII, thi thuc thi khoi code nay  
    .....  
else:  
    Neu khong co exception nao thi thuc thi khoi code nay
```

Dưới đây là một số điểm bạn cần lưu ý:

- Phần code khả nghi mà có khả năng tạo exception cần được bao quanh trong khối try.

- Khối try được theo sau bởi lệnh except. Có thể có một hoặc nhiều lệnh except với một khối try đơn.
- Lệnh except xác định exception mà xảy ra. Trong trường hợp mà exception đó xảy ra, thì lệnh tương ứng được thực thi.
- Ở cuối khối try, bạn có thể cung cấp lệnh else. Nó được thực thi khi không có exception nào xảy ra. Khối else là địa điểm tốt cho code mà không cần sự bảo vệ của khối try.

Ví dụ

Ví dụ sau mở một file, ghi nội dung vào file này và sau đó đóng file, tất cả hoạt động đều thành công:

```
try:  
    fh = open("testfile", "w")  
    fh.write("Day la mot kiem tra nho ve xu ly ngoai le!!")  
except IOError:  
    print "Error: Khong tim thay file"  
else:  
    print "Thanh cong ghi noi dung vao file"  
    fh.close()
```

Kết quả là:

```
Thanh cong ghi noi dung vao file
```

Ví dụ

Ví dụ sau mở một file để ghi trong khi bạn không có quyền ghi, do đó nó sẽ tạo một exception:

```
try:  
    fh = open("testfile", "r")  
    fh.write("Day la mot kiem tra nho ve xu ly ngoai le!!")  
except IOError:
```

```
print "Error: Khong tim thay file"
else:
    print "Thanh cong ghi noi dung vao file"
```

Kết quả là:

```
Error: Khong tim thay file
```

Mệnh đề except mà không xác định Exception trong Python

Lệnh except cũng có thể được sử dụng mà không xác định exception nào. Lệnh try-except này bắt tất cả exception mà xuất hiện. Sử dụng loại lệnh try-except này không phải là sự thực hành lập trình tốt, bởi vì nó bắt tất cả exception nhưng không làm cho lập trình viên biết được cẩn nguyên của vấn đề làm xuất hiện exception đó.

Cú pháp

```
try:
    Ban thuc hien cac hoat dong cua minh tai day;
    Va day la phan code co the tao exception;
    .....
except:
    Neu co bat ky exception nao, thi thuc thi khoi code nay
    .....
else:
    Neu khong co exception nao, thi thuc thi khoi code nay
```

Mệnh đề except với nhiều exception trong Python

Sử dụng cùng lệnh except như trên, bạn có thể khai báo nhiều exception như sau:

```
try:
    Ban thuc hien cac hoat dong cua minh tai day;
    Va day la phan code co the tao exception;
    .....
except(Exception1[, Exception2[,...ExceptionN]]):
    Neu co bat ky exception nao trong danh sach,
```

```
thi thuc thi khoi code nay  
.....  
else:  
Neu khong co exception nao, thi thuc thi khoi code nay
```

Khối try-finally trong Python

Trong trường hợp nếu có bất kỳ code nào mà người dùng muốn được thực thi, dù cho có xuất hiện exception hay không thì khôi code đó có thể được đặt trong khôi finally. Khôi finally sẽ luôn luôn được thực thi bất chấp có hay không exception. Cú pháp của khôi try-finally là:

```
try:  
    Ban thuc hien cac hoat dong cua minh tai day;  
    Va day la phan code co the tao exception;  
    .....  
    Do co exception nen khoi nay bi bo qua  
finally:  
    Khoi nay nen duoc thuc thi  
    .....
```

Ghi chú: Bạn có thể cung cấp một hoặc nhiều mệnh đề except, hoặc một mệnh đề finally, nhưng không được cung cấp cả hai. Ngoài ra bạn cũng không thể sử dụng mệnh đề else với một mệnh đề finally.

Ví dụ

```
try:  
    fh = open("testfile", "w")  
    fh.write("Day la mot kiem tra nho ve xu ly ngoai le!!")  
finally:  
    print "Error: Khong tim thay file"
```

Nếu bạn không có quyền mở file trong chế độ ghi, thì code trên sẽ cho kết quả:

```
Error: Khong tim thay file
```

Ví dụ trên có thể được viết rõ ràng hơn như sau:

```
try:  
    fh = open("testfile", "w")  
    try:  
        fh.write("Day la mot kiem tra nho ve xu ly ngoai le!!")  
    finally:  
        print "Chuan bi dong file"  
        fh.close()  
except IOError:  
    print "Error: Khong tim thay file"
```

Khi một exception được ném trong khối `try`, thì trình thực thi ngay lập tức truyền tới khối `finally`. Sau đó tất cả các lệnh trong khối `finally` được thực thi, exception được tạo lại lần nữa và được xử lý bởi lệnh `except` nếu có mặt trong lớp trên tiếp theo của lệnh `try-except`.

Tham số của một Exception trong Python

Một Exception có thể có một tham số, mà là một giá trị mà cung cấp thông tin bổ sung về vấn đề. Nội dung của tham số là đa dạng tùy vào các exception. Dưới đây là cú pháp:

```
try:  
    Ban thuc hien cac hoat dong cua minh tai day;  
    Va day la phan code co the tao exception;  
    .....  
except Kieu_exception, Tham_so:  
    Ban co the in gia tri cua Tham_so tai day...
```

Nếu bạn viết code trên để xử lý một exception đơn, bạn có thể có một biến theo sau tên của exception trong lệnh `except`. Nếu bạn đang khai báo nhiều exception thì bạn có một biến theo sau tuple của các exception đó.

Ví dụ

Ví dụ của một exception đơn:

```
# Dinh nghia mot ham o day.

def temp_convert(var):
    try:
        return int(var)
    except ValueError, Argument:
        print "Tham so khong chua cac so\n", Argument

# Goi ham tren.

temp_convert("xyz");
```

Tạo một Exception trong Python

Bạn có thể ném tường minh một Exception trong Python bởi sử dụng lệnh raise. Cú pháp của lệnh raise như sau:

```
raise [Exception [, args ]]
```

Ở đây, Lop_Exception là kiểu của exception và tham số value (tùy ý) là một giá trị. Để truy cập giá trị này thì từ khóa as được sử dụng.

Ví dụ

```
try:
    a=10
    print a
    raise NameError("Hello")

except NameError as e:
    print "Mot Exception xuat hien"
    print e
```

Trong ví dụ trên, e được sử dụng như là một biến tham chiếu mà lưu trữ giá trị của exception.

Ghi chú: để bắt một exception, một mệnh đề except phải tham chiếu tới cùng exception đã được ném. Ví dụ, để bắt exception trên, chúng ta phải viết mệnh đề except như sau:

```
try:
```

```
Ban thuc hien cac hoat dong cua minh tai day;  
Va day la phan code co the tao exception;  
...  
except "Invalid level!":  
    Xu ly ngoai le o day...  
else:  
    Phan con lai cua code...
```

Custom Exception trong Python

Python cho phép bạn tạo riêng cho mình các exception bằng cách kế thừa các lớp từ các Standard Exception.

Ví dụ dưới đây liên quan tới RuntimeError. Ở đây, một lớp đã tạo là lớp con của của RuntimeError. Trong khối try, exception được định nghĩa bởi người dùng được tạo và được bắt trong khối except. Biến e được sử dụng để tạo một instance của lớp Networkerror.

```
class Networkerror(RuntimeError):  
    def __init__(self, arg):  
        self.args = arg
```

Sau khi định nghĩa lớp trên, bạn có thể tạo exception như sau:

```
try:  
    raise Networkerror("Bad hostname")  
except Networkerror,e:  
    print e.args
```

Khái niệm hướng đối tượng trong Python

Python là một ngôn ngữ lập trình hướng đối tượng. Do đó với những bạn đã học qua C++ chắc rằng đã khá quen thuộc với các khái niệm về hướng đối tượng này. Chương này sẽ trình bày sơ qua về các thuật ngữ liên quan đến hướng đối tượng cùng với các ví dụ minh họa giúp bạn dễ hiểu hơn về vấn đề đã trình bày.

Một số khái niệm hướng đối tượng

- **Lớp:** Một nguyên mẫu được định nghĩa bởi người dùng cho một đối tượng mà định nghĩa một tập hợp các thuộc tính mà xác định rõ bất kỳ đối tượng nào của lớp đó. Các thuộc tính là các thành viên dữ liệu (các biến class và biến instance) và các phương thức được truy cập thông qua toán tử dot (dấu chấm .).
- **Biến class:** Đây là một biến được chia sẻ bởi tất cả các instance (sự thể hiện) của một lớp. Các biến class được định nghĩa bên trong một lớp nhưng ở bên ngoài bất cứ phương thức nào của lớp đó. Biến class không được sử dụng thường xuyên như biến instance.
- **Thành viên dữ liệu:** Là một biến class hoặc biến instance mà giữ dữ liệu được liên kết với một lớp và các đối tượng của nó.
- **Nạp chồng hàm (overloading):** Là phép gán của nhiều hơn một hành vi tới một hàm cụ thể. Hoạt động được thực hiện là đa dạng do các kiểu của các đối tượng hoặc tham số liên quan.
- **Biến instance:** Là một biến được định nghĩa bên trong một phương thức và chỉ thuộc sở hữu của instance hiện tại của một lớp đó.
- **Tính kế thừa:** Là việc truyền các đặc trưng của một lớp cho các lớp khác mà kế thừa từ lớp ban đầu.
- **Instance:** Là một đối tượng riêng của một lớp nào đó. Một đối tượng obj mà thuộc một lớp Circle là một instance (sự thể hiện) của lớp Circle.
- **Trình thuyết minh:** Là trình tạo một sự thể hiện của một lớp.
- **Phương thức:** Một loại hàm đặc biệt mà được định nghĩa trong một phần định nghĩa lớp.

- **Đối tượng:** Một instance duy nhất của một cấu trúc dữ liệu mà được định nghĩa bởi lớp của nó. Một đối tượng gồm các thành viên dữ liệu (biến class và biến instance) và các phương thức.
- **Nạp chòng toán tử:** Là phép gán của nhiều hơn một hàm cho một toán tử cụ thể.

Tạo các lớp trong Python

Trong Python, lệnh **class** được sử dụng để tạo một lớp mới. Tên của lớp theo ngay sau từ khóa **class** và được theo sau bởi dấu hai chấm, như sau:

```
class TenLop:  
    'Phan documentation string cho lop la tuy y'  
    class_suite
```

- Lớp có một Documentation String mà có thể được truy cập thông qua *TenLop.__doc__*.
- **class_suite** gồm tất cả các lệnh thành phần mà định nghĩa các thành viên lớp, cấu trúc dữ liệu và hàm.

Dưới đây là ví dụ đơn giản về một lớp trong Python:

```
class Sinhvien:  
    'Class co so chung cho tat ca sinh vien'  
    svCount = 0  
  
    def __init__(self, ten, hocphi):  
        self.ten = ten  
        self.hocphi = hocphi  
        Sinhvien(svCount += 1)  
  
    def displayCount(self):  
        print "Tong so Sinh vien %d" % Sinhvien(svCount)  
  
    def displaySinhvien(self):  
        print "Ten : ", self.ten, ", Hoc phi: ", self.hocphi
```

- Biến svCount là một biến class có giá trị được chia sẻ trong tất cả instance của lớp Sinhvien này. Biến này có thể được truy cập dưới dạng Sinhvien(svCount) từ bên trong lớp hoặc bên ngoài lớp.
- Phương thức đầu tiên `__init__()` là một phương thức đặc biệt, là constructor của lớp hoặc phương thức thay thế minh mà Python gọi khi bạn tạo một instance mới của lớp này.
- Bạn khai báo các phương thức khác như các hàm thông thường với exception là tham số đầu tiên cho mỗi phương thức là `self`. Python thêm tham số `self` tới List cho bạn; bạn không cần bao nó khi bạn gọi các phương thức.

Tạo Instance trong Python

Để tạo các instance của một lớp, bạn gọi lớp này bởi sử dụng tên lớp và truyền vào bất kỳ tham số nào mà phương thức `__init__` của nó chấp nhận. Bạn theo dõi ví dụ sau:

```
"Lenh nay tao doi tuong dau tien cua lop Sinhvien"  
sv1 = Sinhvien("Hoang", 4000000)  
  
"Lenh nay tao doi tuong thu hai cua lop Sinhvien"  
sv2 = Sinhvien("Huong", 4500000)
```

Truy cập các thuộc tính trong Python

Bạn truy cập các thuộc tính của đối tượng bởi sử dụng toán tử dot (dấu chấm) với đối tượng. Biến class sẽ được truy cập bởi sử dụng tên lớp như sau:

```
sv1.displaySinhvien()  
sv2.displaySinhvien()  
print "Tong so Sinh vien %d" % Sinhvien(svCount)
```

Bây giờ đặt tất cả khái niệm cùng với nhau:

```
class Sinhvien:  
    'Class co so chung cho tat ca sinh vien'  
    svCount = 0  
  
    def __init__(self, ten, hocphi):
```

```
self.ten = ten
self.hocphi = hocphi
Sinhvien.svCount += 1

def displayCount(self):
    print "Tong so Sinh vien %d" % Sinhvien.svCount

def displaySinhvien(self):
    print "Ten : ", self.ten, ", Hoc phi: ", self.hocphi

"Lenh nay tao doi tuong dau tien cua lop Sinhvien"
sv1 = Sinhvien("Hoang", 4000000)
"Lenh nay tao doi tuong thu hai cua lop Sinhvien"
sv2 = Sinhvien("Huong", 4500000)
sv1.displaySinhvien()
sv2.displaySinhvien()
print "Tong so Sinh vien %d" % Sinhvien.svCount
```

Khi code trên được thực thi sẽ cho kết quả:

```
Ten : Hoang ,Hoc phi: 4000000
Ten : Huong ,Hoc phi: 4500000
Tong so Sinh vien 2
```

Bạn có thể thêm, xóa, hoặc sửa đổi các thuộc tính của các lớp và đối tượng tại bất cứ thời điểm nào.

```
sv1.tuoi = 21 # Them mot thuoc tinh 'tuoi'.
sv1.tuoi = 20 # Sua doi thuoc tinh 'tuoi'.
del sv1.tuoi # Xoa thuoc tinh 'tuoi'.
```

Thay vì sử dụng các lệnh chính thức để truy cập các thuộc tính, bạn có thể sử dụng các hàm sau:

- Hàm **getattr(obj, name[, default])** : Để truy cập thuộc tính của đối tượng.
- Hàm **hasattr(obj, name)** : Để kiểm tra xem một thuộc tính có tồn tại hay không.

- Hàm **setattr(obj, name, value)** : Để thiết lập một thuộc tính. Nếu thuộc tính không tồn tại, thì nó sẽ được tạo.
- Hàm **delattr(obj, name)** : Để xóa một thuộc tính.

Ví dụ:

```
hasattr(sv1, 'tuoi')      # Tra ve true neu thuoc tinh 'tuoi' ton tai  
getattr(sv1, 'tuoi')     # Tra ve gia tri cua thuoc tinh 'tuoi'  
setattr(sv1, 'tuoi', 20)  # Thiet lap thuoc tinh 'tuoi' la 20  
delattr(sv1, 'tuoi')     # Xoa thuoc tinh 'tuoi'
```

Các thuộc tính đã sẵn cho lớp trong Python

Mỗi lớp Python đều giữ các thuộc tính đã được xây dựng sẵn sau và chúng có thể được truy cập bởi sử dụng toán tử dot (dấu chấm .) như bất kỳ thuộc tính khác:

- **__dict__**: Là Dictionary chứa namespace của lớp.
- **__doc__**: Được sử dụng để truy cập Documentation String của lớp nếu có.
- **__name__**: Là tên lớp.
- **__module__**: Là tên Module trong đó lớp được định nghĩa. Thuộc tính là **__main__** trong chế độ tương tác.
- **__bases__**: Là một Tuple chứa các lớp cơ sở.

Với lớp Sinhvien trên, chúng ta sẽ thử truy cập tất cả các thuộc tính này.

```
class Sinhvien:  
    'Class co so chung cho tat ca sinh vien'  
    svCount = 0  
  
    def __init__(self, ten, hocphi):  
        self.ten = ten  
        self.hocphi = hocphi
```

```
Sinhvien.svCount += 1

def displayCount(self):
    print "Tong so Sinh vien %d" % Sinhvien.svCount

def displaySinhvien(self):
    print "Ten : ", self.ten, ", Hoc phi: ", self.hocphi

print "Sinhvien.__doc__:", Sinhvien.__doc__
print "Sinhvien.__name__:", Sinhvien.__name__
print "Sinhvien.__module__:", Sinhvien.__module__
print "Sinhvien.__bases__:", Sinhvien.__bases__
print "Sinhvien.__dict__:", Sinhvien.__dict__
```

Khi code trên được thực thi sẽ cho kết quả:

```
Sinhvien.__doc__: Class co so chung cho tat ca sinh vien
Sinhvien.__name__: Sinhvien
Sinhvien.__module__: __main__
Sinhvien.__bases__: ()
Sinhvien.__dict__: {'__module__': '__main__', 'displayCount':
<function displayCount at 0xb7c84994>, 'svCount': 2,
'displaySinhvien': <function displaySinhvien at 0xb7c8441c>,
'__doc__': 'Class co so chung cho tat ca sinh vien',
'__init__': <function __init__ at 0xb7c846bc>}
```

Hủy đối tượng (Trình dọn rác) trong Python

Python sẽ hủy các đối tượng mà không cần đến nữa (các kiểu đã được xây dựng sẵn hoặc instance của lớp) một cách tự động để giải phóng không gian bộ nhớ. Tiến trình này được gọi là Garbage Collection được thực hiện bởi trình dọn rác Garbage Collector.

Trình dọn rác của Python chạy trong khi thực thi chương trình và được kích hoạt khi số tham chiếu của một đối tượng tiến về 0. Số tham chiếu của một đối tượng thay đổi khi số alias mà trỏ tới nó thay đổi.

Số tham chiếu của một đối tượng tăng khi nó được gán một tên mới hoặc được đặt trong một container (chẳng hạn như List, Tuple, Dictionary). Số tham chiếu của một đối tượng giảm khi nó bị

xóa với lệnh del, tham chiếu của nó được tái gán, hoặc tham chiếu của nó thoát ra khỏi phạm vi. Khi số tham chiếu của một đối tượng tiến về 0, thì Python thu thập nó một cách tự động. Ví dụ:

```
a = 40      # Tao doi tuong <40>
b = a      # Tang so tham chieu cua <40>
c = [b]      # Tang so tham chieu cua <40>

del a      # Giam so tham chieu cua <40>
b = 100    # Giam so tham chieu cua <40>
c[0] = -1  # Giam so tham chieu cua <40>
```

Thường thì bạn sẽ không chú ý khi trình dọn rác hủy một instance và giải phóng bộ nhớ. Nhưng một lớp có thể triển khai phương thức đặc biệt là `__del__()`, được gọi là một destructor, mà được triệu hồi khi instance là chuẩn bị được hủy. Phương thức này có thể được sử dụng để xóa bất kỳ nguồn bộ nhớ nào được sử dụng bởi một instance.

Ví dụ

`__del__()` destructor này in tên lớp của một instance mà chuẩn bị được hủy.

```
class Point:
    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y
    def __del__(self):
        class_name = self.__class__.__name__
        print(class_name, "destroyed")

pt1 = Point()
pt2 = pt1
pt3 = pt1
print(id(pt1), id(pt2), id(pt3)) # in id cua doi tuong
del pt1
del pt2
```

```
del pt3
```

Khi code trên được thực thi sẽ cho kết quả sau:

```
3083401324 3083401324 3083401324
```

```
Point destroyed
```

Ghi chú: Một cách lý tưởng nhất là bạn nên định nghĩa các lớp của bạn trong file riêng biệt, sau đó bạn nên import chúng trong file chương trình chính bởi sử dụng lệnh **import**.

Kế thừa lớp trong Python

Thay vì bắt đầu viết code cho một lớp mới, bạn có thể tạo một lớp bằng việc kế thừa nó từ một lớp đã tồn tại trước đó bằng cách liệt kê lớp cha trong cặp dấu ngoặc đơn sau tên lớp mới.

Lớp con kế thừa các thuộc tính của lớp cha của nó, và bạn có thể sử dụng các thuộc tính như thể là chúng đã được định nghĩa trong lớp con đó. Một lớp con cũng có thể ghi đè các thành viên dữ liệu và các phương thức từ lớp cha.

Cú pháp

Các lớp kế thừa được khai báo khá giống như lớp cha của nó; tuy nhiên, một danh sách lớp cơ sở để kế thừa từ đó được cung cấp sau tên lớp mới.

```
class Tenlopcon (LopCha1[, LopCha2, ...]):  
    'Phan documentation string cua Class la tuy y'  
    class_suite
```

Ví dụ

```
class Parent:          # dinh nghia lop cha  
    parentAttr = 100  
  
    def __init__(self):  
        print "Goi constructor cua lop cha"  
  
    def parentMethod(self):  
        print 'Goi phuong thuc cua lop cha'
```

```
def setattr(self, attr):
    Parent.parentAttr = attr

def getattr(self):
    print "Thuoc tinh cua lop cha :", Parent.parentAttr

class Child(Parent): # dinh nghia lop con
    def __init__(self):
        print "Goi constructor cua lop con"

    def childMethod(self):
        print 'Goi phuong thuc cua lop con'

c = Child()          # instance cua lop con
c.childMethod()      # lop con goi phuong thuc cua no
c.parentMethod()    # goi phuong thuc cua lop cha
c.setAttr(200)       # tiep tuc goi phuong thuc cua lop cha
c.getAttr()          # tiep tuc goi phuong thuc cua lop cha
```

Khi code trên được thực thi sẽ cho kết quả sau:

```
Goi constructor cua lop con
Goi phuong thuc cua lop con
Goi phuong thuc cua lop cha
Thuoc tinh cua lop cha : 200
```

Theo cách tương tự, bạn có thể kế thừa một lớp từ nhiều lớp cha như sau:

```
class A:          # dinh nghia lop A
    .....

class B:          # dinh nghia lop B
    .....

class C(A, B):   # lop con cua A va B
```

.....

Bạn có thể sử dụng các hàm `issubclass()` hoặc `isinstance()` để kiểm tra mối quan hệ của hai lớp và `instance`.

- Hàm **issubclass(sub, sup)** trả về true nếu lớp con **sub** đã cho thực sự là lớp con của lớp cha **sup**.
- Hàm **isinstance(obj, Class)** trả về true nếu **obj** là một instance của lớp **Class** hoặc là một instance của lớp con của **Class**.

Ghi đè phương thức trong Python

Bạn có thể ghi đè các phương thức của lớp cha. Một trong các lý do để thực hiện việc ghi đè phương thức của lớp cha là bạn muốn có tính năng khác biệt hoặc đặc biệt trong lớp con.

Ví dụ

```
class Parent:          # dinh nghia lop cha
    def myMethod(self):
        print 'Goi phuong thuc cua lop cha'

class Child(Parent): # dinh nghia lop con
    def myMethod(self):
        print 'Goi phuong thuc cua lop con'

c = Child()          # instance cua lop con
c.myMethod()         # lop con goi phuong thuc duoc ghi de
```

Kết quả là:

```
Goi phuong thuc cua lop con
```

Nạp chồng phương thức trong Python

Bảng dưới đây liệt kê một số tính năng chung mà bạn có thể ghi đè trong các lớp riêng của bạn.

STT	Phương thức, Miêu tả và Lời gọi mẫu
1	<code>__init__(self [,args...])</code> Là constructor (với bất kỳ tham số tùy ý nào) Lời gọi mẫu : <code>obj = tenLop(args)</code>
2	<code>__del__(self)</code> Là destructor, xóa một đối tượng Lời gọi mẫu : <code>del obj</code>
3	<code>__repr__(self)</code> Biểu diễn chuỗi có thể ước lượng Lời gọi mẫu : <code>repr(obj)</code>
4	<code>__str__(self)</code> Biểu diễn chuỗi có thể in được Lời gọi mẫu : <code>str(obj)</code>
5	<code>__cmp__(self, x)</code> So sánh đối tượng Lời gọi mẫu : <code>cmp(obj, x)</code>

Nạp chồng toán tử trong Python

Giả sử bạn đã tạo một lớp Vector để biểu diễn các vector hai chiều. Điều gì xảy ra khi bạn sử dụng toán tử cộng (+) để cộng chúng? Có thể nói vui rằng, lúc đó Python sẽ la hét vào mặt bạn.

Tuy nhiên, bạn có thể định nghĩa phương thức **`__add__`** trong lớp của bạn để thực hiện phép cộng vector và sau đó phép cộng vector sẽ vận hành như bạn mong đợi.

Ví dụ

```
class Vector:  
    def __init__(self, a, b):  
        self.a = a
```

```
self.b = b

def __str__(self):
    return 'Vector (%d, %d)' % (self.a, self.b)

def __add__(self,other):
    return Vector(self.a + other.a, self.b + other.b)

v1 = Vector(2,10)
v2 = Vector(5,-2)
print v1 + v2
```

Kết quả là:

```
Vector(7,8)
```

Ẩn dữ liệu (Data Hiding) trong Python

Các thuộc tính của một đối tượng có thể hoặc không thể là nhìn thấy với bên ngoài phần định nghĩa lớp. Bạn cần đặt tên các thuộc tính với một tiền tố là hai dấu gạch dưới, và sau đó các thuộc tính này sẽ không là nhìn thấy với bên ngoài.

Ví dụ

```
class JustCounter:

    __secretCount = 0

    def count(self):
        self.__secretCount += 1
        print self.__secretCount

counter = JustCounter()
counter.count()
counter.count()
```

```
print counter.__secretCount
```

Kết quả là:

```
1  
2  
Traceback (most recent call last):  
  File "test.py", line 12, in <module>  
    print counter.__secretCount  
AttributeError: JustCounter instance has no attribute '__secretCount'
```

Python bảo vệ các thành viên đó bằng cách thay đổi nội tại tên để bao tên lớp. Bạn có thể truy cập các thuộc tính này dưới dạng như **doi_tuong._tenLop__tenThuocTinh**. Nếu bạn thay thế dòng cuối cùng như sau, thì nó sẽ làm việc cho bạn.

```
.....  
print counter._JustCounter__secretCount
```

Kết quả là:

```
1  
2  
2
```

Regular Expression trong Python

Một Regular Expression là một dãy ký tự đặc biệt giúp bạn so khớp hoặc tìm các chuỗi khác hoặc tập hợp các chuỗi, bởi sử dụng một cú pháp riêng trong một pattern. Regular Expression được sử dụng phổ biến trong thế giới UNIX.

re Module cung cấp sự hỗ trợ đầy đủ các Perl-like Regular Expression trong Python. Module này tạo Exception là **re.error** nếu xảy ra một lỗi trong khi biên dịch hoặc khi sử dụng một Regular Expression.

Có hai hàm quan trọng sẽ được sử dụng để xử lý Regular Expression, đó là:

Hàm match trong Python

Hàm này cố gắng so khớp pattern với string với các **flag** tùy ý. Dưới đây là cú pháp cho hàm này.

```
re.match(pattern, string, flags=0)
```

Chi tiết về tham số:

- **pattern** : Đây là Regular Expression để được so khớp.
- **string** : Đây là chuỗi, mà sẽ được tìm kiếm để so khớp pattern tại phần đầu của chuỗi.
- **flags** : Bạn có thể xác định các flag khác nhau bởi sử dụng toán tử |. Các modifier này sẽ được liệt kê ở bảng bên dưới.

Hàm `re.match` trả về một đối tượng **match** nếu thành công và trả về **None** nếu thất bại. Chúng ta sử dụng hàm `group(num)` hoặc `groups()` của đối tượng `match` để lấy biểu thức đã được so khớp (kết nối).

- Phương thức `group(num=0)` trả về toàn bộ kết nối (hoặc num phân nhóm cụ thể).
- Phương thức `groups()` trả về tất cả các phân nhóm kết nối trong một Tuple (là trống nếu không có kết nối hay so khớp nào).

Ví dụ:

```
import re
```

```
line = "Hoc Python la de hon hoc Java?"\n\nmatchObj = re.match( r'(.*) la (..*?) .*', line, re.M|re.I)\n\nif matchObj:\n    print "matchObj.group() : ", matchObj.group()\n    print "matchObj.group(1) : ", matchObj.group(1)\n    print "matchObj.group(2) : ", matchObj.group(2)\nelse:\n    print "Khong co ket noi!!"
```

Kết quả là:

```
matchObj.group() : Hoc Python la de hon hoc Java?\nmatchObj.group(1) : Hoc Python\nmatchObj.group(2) : de
```

Hàm search trong Python

Hàm này tìm kiếm cho sự xuất hiện đầu tiên của pattern bên trong string với các flags tùy ý. Dưới đây là cú pháp cho hàm search:

```
re.search(pattern, string, flags=0)
```

Các tham số được giải thích như trong hàm match.

Hàm re.search trả về một đối tượng match nếu thành công và trả về None nếu thất bại. Chúng ta sử dụng hàm group(num) hoặc groups() của đối tượng match để lấy biểu thức đã được so khớp (kết nối). Các hàm này đã được trình bày ở trên.

Ví dụ

```
import re\n\nline = "Hoc Python la de hon hoc Java?";\n\nsearchObj = re.search( r'(.*) la (..*?) .*', line, re.M|re.I)
```

```
if searchObj:  
    print "searchObj.group() : ", searchObj.group()  
    print "searchObj.group(1) : ", searchObj.group(1)  
    print "searchObj.group(2) : ", searchObj.group(2)  
else:  
    print "Khong tim thay!!"
```

Kết quả là:

```
searchObj.group() : Hoc Python la de hon hoc Java?  
searchObj.group(1) : Hoc Python  
searchObj.group(2) : de
```

Phân biệt match và search trong Python

Python cung cấp hai hoạt động cơ sở dựa trên Regular Expression, đó là: **match** để kiểm tra chỉ một kết nối tại phần đầu của chuỗi, trong khi **search** tìm kiếm một kết nối ở bất cứ đâu trong chuỗi.

Ví dụ

```
import re  
  
line = "Hoc Python la de hon hoc Java?";  
  
matchObj = re.match( r'thon', line, re.M|re.I)  
if matchObj:  
    print "match --> matchObj.group() : ", matchObj.group()  
else:  
    print "Khong co ket noi!!"  
  
searchObj = re.search( r'thon', line, re.M|re.I)  
if searchObj:  
    print "search --> searchObj.group() : ", searchObj.group()  
else:
```

```
print "Khong tim thay!!"
```

Kết quả là:

```
Khong co ket noi!!
search --> searchObj.group() :  thon
```

Tìm kiếm và thay thế trong Python

Một trong những phương thức quan trọng nhất mà sử dụng với Regular Expression là **sub**. Dưới đây là cú pháp:

```
re.sub(pattern, repl, string, max=0)
```

Phương thức này thay thế tất cả sự xuất hiện của pattern trong string với repl. Phương thức này sẽ thay thế tất cả sự xuất hiện trừ khi bạn cung cấp tham số max. Phương thức này trả về chuỗi đã được sửa đổi.

Ví dụ

```
import re

phone = "01633-810-628 # Day la so dien thoai"

# Xoa cac comment
num = re.sub(r'#.*$', "", phone)
print "So dien thoai : ", num

# Xoa cac ky tu khong phai ky so
num = re.sub(r'\D', "", phone)
print "So dien thoai : ", num
```

Kết quả là:

```
So dien thoai :  01633-810-628
So dien thoai :  01633810628
```

Danh sách modifier trong Python

Các modifier được liệt kê dưới đây có thể được sử dụng như là các flag tùy ý cho các hàm match và search. Bạn có thể cung cấp nhiều modifier bởi sử dụng toán tử |.

Modifier	Miêu tả
re.I	Thực hiện việc kết nối hoặc so khớp không phân biệt kiểu chữ
re.L	Thông dịch các từ theo Locale hiện tại
re.M	Làm cho \$ kết nối với phần cuối của một dòng (mà không chỉ kết nối phần cuối của chuỗi) và làm cho ^ kết nối với phần đầu của bất cứ dòng nào (mà không chỉ kết nối phần đầu của chuỗi)
re.S	Làm cho dot (dấu chấm) kết nối với bất kỳ ký tự nào, bao gồm một newline
re.U	Thông dịch các chữ cái theo bộ ký tự Unicode. Flag này ảnh hưởng tới hành vi của \w, \W, \b, \B
re.X	Cho phép cú pháp "cuter" regular expression. Nó bỏ qua các khoảng trắng whitespace (ngoại trừ bên trong một [] hoặc khi được tránh bởi \) và coi # dưới dạng một comment

Các Pattern trong Python

Ngoại trừ các ký tự điều khiển, (+ ? . * ^ \$ () [] {} | \), thì tất cả ký tự còn lại sẽ kết nối với chính chúng. Bạn có thể tránh một ký tự điều khiển bằng cách đặt trước nó một dấu \.

Dưới đây là danh sách các pattern có sẵn trong Python:

Pattern	Miêu tả
^	Dưới đây là danh sách các pattern có sẵn trong Python:

\$	Kết nối với phần cuối của dòng
.	Kết nối bất kỳ ký tự đơn nào ngoại trừ newline. Sử dụng tùy chọn m cho phép nó kết nối với newline
[...]	Kết nối với bất kỳ ký tự đơn nào trong []
[^...]	Kết nối với bất kỳ ký tự đơn nào không ở trong []
re*	Kết nối với 0 hoặc nhiều sự xuất hiện của biểu thức đặt trước
re+	Kết nối với 1 hoặc nhiều sự xuất hiện của biểu thức đặt trước
re?	Kết nối với 0 hoặc 1 sự xuất hiện của biểu thức đặt trước
re{ n}	Kết nối với n sự xuất hiện của biểu thức đặt trước
re{ n,}	Kết nối với n hoặc nhiều hơn sự xuất hiện của biểu thức đặt trước
re{ n, m}	Kết nối với ít nhất n và nhiều nhất m sự xuất hiện của biểu thức đặt trước
a b	Kết nối với a hoặc b
(re)	Nhóm các Regular Expression và ghi nhớ text đã kết nối
(?imx)	Bật toggle tạm thời các tùy chọn i, m hoặc x bên trong một Regular Expression. Nếu trong cặp dấu ngoặc đơn thì chỉ khu vực đó bị ảnh hưởng
(?-imx)	Tắt toggle tạm thời các tùy chọn i, m hoặc x bên trong một Regular Expression. Nếu trong cặp dấu ngoặc đơn thì chỉ khu vực đó bị ảnh

	hướng
(?: re)	Nhóm các Regular Expression mà không ghi nhớ text đã kết nối
(?imx: re)	Bật toggle tạm thời các tùy chọn i, m, hoặc x bên trong cặp dấu ngoặc đơn
(?-imx: re)	Tắt toggle tạm thời các tùy chọn i, m, hoặc x bên trong cặp dấu ngoặc đơn
(?#...)	Comment
(?= re)	Xác định vị trí bởi sử dụng một pattern. Không có một dãy giá trị
(?! re)	Xác định vị trí bởi sử dụng sự phủ định pattern. Không có một dãy giá trị
(?> re)	Kết nối pattern độc lập mà không backtrack
\w	Kết nối các ký tự từ
\W	Kết nối các ký tự không phải là từ
\s	Kết nối với whitespace. Tương đương với [\t\n\r\f]
\S	Kết nối với các ký tự không là whitespace
\d	Kết nối với các ký số. Tương đương với [0-9]
\D	Kết nối với các ký tự không là ký số

\A	Kết nối với phần đầu của chuỗi
\Z	Kết nối với phần cuối của chuỗi. Nếu một newline tồn tại, nó kết nối với phần ở trước newline
\z	Kết nối với phần cuối của chuỗi
\G	Kết nối với điểm mà tại đó kết nối cuối cùng được tìm thấy
\b	Kết nối với các giới hạn từ khi bên ngoài các dấu []. Kết nối với backspace (mã là 0x08) khi bên trong các dấu []
\B	Kết nối với các giới hạn không phải là từ
\n, \t, etc.	Kết nối với newline, carriage return, tab, ...
\1...\ 9	Kết nối với biểu thức con được nhóm thứ n
\10	Kết nối biểu thức con được nhóm thứ n nếu nó đã kết nối. Nếu không, tham chiếu tới biểu diễn bát phân của một mã ký tự

Ví dụ lớp ký tự trong Python

Ví dụ	Miêu tả
[Pp]ython	Kết nối với "Python" hoặc "python"
rub[ye]	Kết nối với "ruby" hoặc "rube"
[aeiou]	Kết nối với bất kỳ một nguyên âm nào

[0-9]	Kết nối với bất kỳ ký số nào, giống dạng [0123456789]
[a-z]	Kết nối với bất kỳ chữ cái thường ASCII nào
[A-Z]	Kết nối với bất kỳ chữ cái hoa ASCII nào
[a-zA-Z0-9]	Kết nối với bất kỳ cái nào ở trên
[^aeiou]	Kết nối với bất kỳ cái nào ở trên ngoại trừ nguyên âm
[^0-9]	Kết nối với bất kỳ cái nào ở trên ngoại trừ một ký số

Lập trình CGI trong Python

CGI (là viết tắt của Common Gateway Interface) là một tập hợp các chuẩn mà định nghĩa cách thông tin được trao đổi giữa Web Server và một Custom Script. Phiên bản CGI hiện tại là CGI/1.1.

Cấu hình Web Server

Trước khi tiến hành lập trình CGI, bạn đảm bảo rằng Web Server của bạn hỗ trợ CGI và được cấu hình để xử lý các chương trình CGI. Tất cả chương trình CGI được thực thi bởi HTTP đều được giữ trong một thư mục đã được cấu hình trước.

Thư mục này được gọi là CGI Directory và theo qui ước nó được đặt tên dạng /var/www/cgi-bin. Các CGI file có đuôi là .cgi, nhưng bạn cũng có thể giữ các file trong .py.

Theo mặc định, Linux Server được cấu hình để chạy các script trong thư mục cgi-bin trong /var/www. Nếu bạn muốn xác định bất kỳ thư mục nào khác để chạy CGI script của bạn, thì bạn comment các dòng sau trong httpd.conf file:

```
<Directory "/var/www/cgi-bin">
    AllowOverride None
    Options ExecCGI
    Order allow,deny
    Allow from all
</Directory>

<Directory "/var/www/cgi-bin">
    Options All
</Directory>
```

Chương trình CGI đầu tiên

Dưới đây là một CGI script có tên là hello.py, được giữ trong thư mục /var/www/cgi-bin. Trước khi chạy, bạn cần thay đổi mode của file này bởi sử dụng chmod 755 hello.py để làm file này có thể thực thi.

```
print "Content-type:text/html\r\n\r\n"
```

```
print '<html>'  
print '<head>'  
print '<title>Lap trinh CGI trong Python</title>'  
print '</head>'  
print '<body>'  
print '<h2>Day la chuong trinh CGI dau tien trong Python</h2>'  
print '</body>'  
print '</html>'
```

Kết quả là:

Day la chuong trinh CGI dau tien trong Python

Đây là một Python script đơn giản để viết kết quả trên STDOUT chuẩn là màn hình. Dòng **Content-type:text/html\r\n\r\n** được gửi trả lại trình duyệt và nó xác định kiểu nội dung để được hiển thị trên màn hình trình duyệt.

HTTP Header

Dòng **Content-type:text/html\r\n\r\n** là một phần của HTTP Header mà được gửi tới trình duyệt để giúp trình duyệt hiểu nội dung cần được hiển thị. Tất cả HTTP Header sẽ là trong form sau:

HTTP Ten Truong: Noi Dung Cua Truong

Vi du

Content-type: text/html\r\n\r\n

Dưới đây là một số HTTP Header quan trọng khác mà bạn sẽ sử dụng thường xuyên trong lập trình CGI:

Header	Miêu tả
Content-type:	Một chuỗi MIME định nghĩa định dạng của file được trả về. Ví dụ Content-type:text/html
Expires: Date	Ngày mà thông tin trả về hết hiệu lực. Nó được sử dụng bởi trình duyệt để xác định khi nào trang cần được refresh. Một chuỗi

	date hợp lệ là trong định dạng 01 Jan 1998 12:00:00 GMT.
Location: URL	URL mà được trả về thay cho URL đã được yêu cầu. Bạn có thể sử dụng trường này để chuyển hướng một yêu cầu tới bất kỳ file nào
Last-modified: Date	Ngày sửa đổi cuối cùng của nguồn
Content-length: N	Độ dài (số byte) của dữ liệu đang được trả về. Trình duyệt sử dụng giá trị này để báo cáo thời gian download ước lượng cho một file
Set-Cookie: String	Thiết lập cookie được truyền thông qua <i>String</i>

Các biến môi trường của CGI

Tất cả chương trình CGI có quyền truy cập tới các biến môi trường sau. Các biến này đóng một vai trò quan trọng trong khi viết bất cứ chương trình CGI nào.

Tên biến	Miêu tả
CONTENT_TYPE	Kiểu dữ liệu của nội dung. Được sử dụng khi Client đang gửi nội dung đính kèm tới Server. Ví dụ: file upload
CONTENT_LENGTH	Độ dài của thông tin truy vấn. Chỉ có sẵn cho các yêu cầu POST
HTTP_COOKIE	Trả về các Cookie đã thiết lập trong dạng là cặp key/value
HTTP_USER_AGENT	Trường User-Agent chứa thông tin về user agent tạo yêu cầu. Đây là tên của trình duyệt web
PATH_INFO	Path cho CGI script

QUERY_STRING	Thông tin mã hóa URL được gửi với phương thức GET
REMOTE_ADDR	Địa chỉ IP của host từ xa mà tạo yêu cầu. Biến này hữu ích cho log và xác nhận
REMOTE_HOST	Tên đầy đủ của host tạo yêu cầu. Nếu thông tin này không có sẵn, thì REMOTE_ADDR có thể được sử dụng để lấy địa chỉ IP
REQUEST_METHOD	Phương thức được sử dụng để tạo yêu cầu. Phương thức được sử dụng phổ biến là GET và POST
SCRIPT_FILENAME	Path đầy đủ tới CGI script
SCRIPT_NAME	Tên của CGI script
SERVER_NAME	Tên của CGI script
SERVER_SOFTWARE	Tên và phiên bản của phần mềm mà Server đang chạy trên đó

Chương trình CGI sau sẽ liệt kê tất cả các biến CGI.

```
import os

print "Content-type: text/html\r\n\r\n";
print "<font size=+1>Environment</font><br>";
for param in os.environ.keys():
    print "<b>%20s</b>: %s<br>" % (param, os.environ[param])
```

Truyền thông tin bởi sử dụng phương thức GET

Phương thức GET gửi thông tin người dùng đã mã hóa được phụ thêm tới yêu cầu trang. Trang và thông tin mã hóa được phân biệt bởi ký tự ? như sau:

http://www.test.com/cgi-bin/hello.py?key1=value1&key2=value2

Phương thức GET là phương thức mặc định để truyền thông tin từ trình duyệt tới Web Server và nó tạo ra một chuỗi dài xuất hiện trong Location:box của trình duyệt.

Nếu bạn có password hoặc bất cú thông tin nhạy cảm nào khác cần truyền tới Server thì bạn đừng bao giờ sử dụng phương thức GET. Phương thức GET có giới hạn kích cỡ: chỉ có 1024 ký tự có thể được gửi trong một chuỗi yêu cầu. Phương thức GET gửi thông tin bởi QUERY_STRING Header và sẽ là có thể truy cập trong chương trình CGI thông qua biến môi trường QUERY_STRING.

Bạn có thể truyền thông tin bằng cách đơn giản là nối chuỗi các cặp key và value cùng với bất cứ URL nào hoặc bạn có thể sử dụng thẻ form trong HTML.

Phương thức GET: Ví dụ URL đơn giản

URL đơn giản sau sẽ truyền hai giá trị tới chương trình hello_get.py bởi sử dụng phương thức GET.

/cgi-bin/hello_get.py?first_name=HOANG&last_name=NGUYEN

Dưới đây là hello_get.py để xử lý đầu vào đã được cung cấp bởi trình duyệt web. Chúng ta đang sử dụng **cgi** Module giúp cho việc truy cập thông tin đã truyền được dễ dàng hơn.

```
# Import cac module de xu ly CGI
import cgi, cgitb

# Tao instance cua FieldStorage
form = cgi.FieldStorage()

# Lay du lieu tu cac truong
first_name = form.getvalue('first_name')
last_name = form.getvalue('last_name')

print "Content-type:text/html\r\n\r\n"
```

```
print "<html>"  
print "<head>"  
print "<title>Chuong trinh CGI thu hai</title>"  
print "</head>"  
print "<body>"  
print "<h2>Hello %s %s</h2>" % (first_name, last_name)  
print "</body>"  
print "</html>"
```

Phương thức GET: Ví dụ FORM đơn giản

Ví dụ sau sẽ truyền hai giá trị bởi sử dụng HTML form và nút submit. Chúng ta sử dụng hello_get.py giống như trên để xử lý đầu vào này.

```
<form action="/cgi-bin/hello_get.py" method="get">  
First Name: <input type="text" name="first_name"> <br />  
  
Last Name: <input type="text" name="last_name" />  
<input type="submit" value="Submit" />  
</form>
```

Truyền thông tin bởi sử dụng phương thức POST

Một phương thức đáng tin cậy hơn để truyền thông tin tới một chương trình CGI là phương thức POST. Phương thức này đóng gói thông tin theo đúng như cách của phương thức GET, nhưng thay vì gửi nó dưới dạng một chuỗi text sau một dấu ? trong URL, thì nó gửi dưới dạng một thông điệp riêng rẽ. Thông điệp này vào trong CGI script trong dạng đầu vào chuẩn.

Ví dụ sau cũng sử dụng hello_get.py ở trên.

```
# Import cac module de xu ly CGI  
import cgi, cgitb  
  
# Tao instance cua FieldStorage  
form = cgi.FieldStorage()
```

```
# Lay du lieu tu cac truong
first_name = form.getvalue('first_name')
last_name = form.getvalue('last_name')

print "Content-type:text/html\r\n\r\n"
print "<html>"
print "<head>"
print "<title>Chuong trinh CGI thu hai</title>"
print "</head>"
print "<body>"
print "<h2>Hello %s %s</h2>" % (first_name, last_name)
print "</body>"
print "</html>"
```

Sử dụng lại ví dụ trên để truyền hai giá trị bởi sử dụng HTML form và nút submit.

```
<form action="/cgi-bin/hello_get.py" method="post">
First Name: <input type="text" name="first_name"><br />
Last Name: <input type="text" name="last_name" />

<input type="submit" value="Submit" />
</form>
```

Truyền Checkbox tới chương trình CGI

HTML code sau là cho một form với hai checkbox:

```
<form action="/cgi-bin/checkbox.cgi" method="POST" target="_blank">
<input type="checkbox" name="toan" value="on" /> Toan
<input type="checkbox" name="vatly" value="on" /> VatLy
<input type="submit" value="Chon Mon Hoc" />
</form>
```

Dưới đây là checkbox.cgi để xử lý đầu vào được cung cấp bởi trình duyệt web:

```
# Import cac module de xu ly CGI
import cgi, cgitb

# Tao instance cua FieldStorage
form = cgi.FieldStorage()

# Lay du lieu tu cac truong
if form.getvalue('toan'):
    toan_flag = "ON"
else:
    toan_flag = "OFF"

if form.getvalue('vatly'):
    vatly_flag = "ON"
else:
    vatly_flag = "OFF"

print "Content-type:text/html\r\n\r\n"
print "<html>"
print "<head>"
print "<title>Vi du Checkbox</title>"
print "</head>"
print "<body>"
print "<h2> Mon Toan la : %s</h2>" % toan_flag
print "<h2> Mon Vat Ly la : %s</h2>" % vatly_flag
print "</body>"
print "</html>"
```

Truyền RadioButton tới chương trình CGI

HTML code sau cho một form với hai Radiobutton:

```
<form action="/cgi-bin/radiobutton.py" method="post" target="_blank">
```

```
<input type="radio" name="subject" value="toan" /> Toan  
<input type="radio" name="subject" value="vatly" /> VatLy  
<input type="submit" value="Chon Mon Hoc" />  
</form>
```

Và đây là radiobutton.py để xử lý đầu vào được cung cấp bởi trình duyệt web:

```
# Import cac module de xu ly CGI  
import cgi, cgitb  
  
# Tao instance cua FieldStorage  
form = cgi.FieldStorage()  
  
# Lay du lieu tu cac truong  
if form.getvalue('subject'):  
    subject = form.getvalue('subject')  
else:  
    subject = "Khong duoc thiet lap"  
  
print "Content-type:text/html\r\n\r\n"  
print "<html>"  
print "<head>"  
print "<title>Vi du Radio button</title>"  
print "</head>"  
print "<body>"  
print "<h2> Mon hoc ban da chon la %s</h2>" % subject  
print "</body>"  
print "</html>"
```

Ví dụ File Upload

Để upload một file, HTML form phải có thuộc tính enctype được thiết lập thành **multipart/form-data**.

```
<html>
<body>
<form enctype="multipart/form-data"
        action="save_file.py" method="post">
<p>File: <input type="file" name="filename" /></p>
<p><input type="submit" value="Upload" /></p>
</form>
</body>
</html>
```

Dưới đây là save_file.py để xử lý file upload:

```
import cgi, os
import cgitb; cgitb.enable()

form = cgi.FieldStorage()

# Lay ten file o day.
fileitem = form['filename']

# Kiem tra xem file da duoc upload chua
if fileitem.filename:
    fn = os.path.basename(fileitem.filename)
    open('/tmp/' + fn, 'wb').write(fileitem.file.read())

    message = 'File co ten la "' + fn + '" duoc upload thanh cong'

else:
    message = 'Khong co file nao duoc upload'

print """
Content-Type: text/html\n
```

```
<html>
<body>
    <p>%s</p>
</body>
</html>
"""
% (message,)
```

Sử dụng Cookie

Cookie là bản ghi dữ liệu thuần text của 5 trường biến sau:

- **Expires:** Ngày cookie sẽ hết hạn. Nếu là trống, thì cookie sẽ hết hạn khi khách truy cập thoát khỏi trình duyệt.
- **Domain:** Tên miền của site của bạn.
- **Path:** Path tới thư mục hoặc trang web mà thiết lập cookie. Nó có thể là trống nếu bạn muốn thu nhận cookie từ bất kỳ thư mục hoặc trang nào.
- **Secure:** Nếu trường này chứa từ secure, thì cookie có thể chỉ được thu nhận bởi một server an toàn. Nếu để trống, thì không tồn tại giới hạn nào.
- **Name=Value:** Cookie được thiết lập và thu nhận trong dạng các cặp key-value.

Thiết lập cookie

Để gửi cookie tới trình duyệt là khá dễ dàng. Các cookie này được gửi cùng với HTTP Header, ở trước trường Content-type. Giả sử bạn muốn thiết lập UserID và Password là các cookie, thì việc này được thực hiện như sau:

```
print "Set-Cookie:UserID=XYZ;\r\n"
print "Set-Cookie:Password=XYZ123;\r\n"
print "Set-Cookie:Expires=Tuesday, 31-Nov-2015 23:12:40 GMT;\r\n"
print "Set-Cookie:Domain=www.vietjack.com;\r\n"
print "Set-Cookie:Path=/perl;\r\n"
print "Content-type:text/html\r\n\r\n"
```

.....Rest of the HTML Content....

Qua ví dụ trên, bạn có thể thấy rằng chúng ta đã sử dụng trường **Set-Cookie** để thiết lập các cookie. Việc thiết lập các thuộc tính của cookie như Expires, Domain, Path là tùy ý. Bạn cần chú ý là các cookie được thiết lập trước khi gửi trường "**Content-type:text/html\r\n\r\n**".

Thu nhận Cookie

Để thu nhận tất cả các Cookie đã thiết lập là khá dễ dàng. Các cookie được lưu trữ trong biến môi trường HTTP_COOKIE của CGI và có dạng sau:

key1=value1;key2=value2;key3=value3....

Dưới đây là ví dụ đơn giản minh họa cách thu nhận các cookie:

```
# Import cac module de xu ly CGI
from os import environ
import cgi, cgitb

if environ.has_key('HTTP_COOKIE'):
    for cookie in map(strip, split(environ['HTTP_COOKIE'], ';')):
        (key, value) = split(cookie, '=')
        if key == "UserID":
            user_id = value

        if key == "Password":
            password = value

print "Ten dang nhap = %s" % user_id
print "Mat khau = %s" % password
```

Truy cập MySQL Database trong Python

Chương này sẽ giới thiệu khái quát cho bạn về cách truy cập Database bởi sử dụng Python và giới thiệu qua về một số hoạt động cơ bản trên Database như INSERT, UPDATE, DELETE ...

Trước khi theo dõi và thực hành chương này, bạn cần phải tải một DB API Module riêng cho mỗi Database bạn cần truy cập. DB API cung cấp một chuẩn tối thiểu để làm việc với Database bởi sử dụng Python. Trong chương này, chúng ta sẽ sử dụng MySQL, vì thế trước hết chúng ta hãy tìm hiểu qua về MySQLdb Module.

MySQLdb là gì?

MySQLdb là một Interface để kết nối tới một MySQL Database Server từ Python. Nó triển khai Python Database API 2.0 và được xây dựng trên cùng của MySQL C API.

Cách cài đặt MySQLdb?

Trước khi tiến hành, bạn cần cài đặt MySQL trên thiết bị của bạn. Sau đó gõ dòng Python script sau và thực thi nó:

```
import MySQLdb
```

Nếu nó cho kết quả sau, thì nghĩa là MySQLdb Module đã không được cài đặt:

```
Traceback (most recent call last):
  File "test.py", line 3, in <module>
    import MySQLdb
ImportError: No module named MySQLdb
```

Để cài đặt MySQLdb Module, tải nó từ [MySQLdb Download](#) và tiến hành như sau:

```
$ gunzip MySQL-python-1.2.2.tar.gz
$ tar -xvf MySQL-python-1.2.2.tar
$ cd MySQL-python-1.2.2
$ python setup.py build
$ python setup.py install
```

Kết nối Database trong Python

Trước khi kết nối với một MySQL Database, đảm bảo:

- Bạn đã tạo một Database có tên là TESTDB.
- Bạn đã tạo một bảng là SINHVIEN trong TESTDB.
- Bảng này có các trường HO, TEN, TUOI, GIOITINH và HOCPHI
- User ID là testuser và password là test123 được thiết lập để truy cập TESTDB.
- MySQLdb Module được cài đặt phù hợp trên thiết bị của bạn.
- Bạn đã hiểu cơ bản về MySQL, nếu chưa, bạn có thể tham khảo [Bài hướng dẫn MySQL](#).

Dưới đây là ví dụ về kết nối với TESTDB.

```
import MySQLdb

# mo ket noi voi Database
db = MySQLdb.connect("localhost","testuser","test123","TESTDB" )

# chuan bi mot doi tuong cursor boi su dung phuong thuc cursor()
cursor = db.cursor()

# Thuc thi truy van SQL boi su dung phuong thuc execute().
cursor.execute("SELECT VERSION()")

# Lay mot hang boi su dung phuong thuc fetchone().
data = cursor.fetchone()

print "Database version : %s " % data

# ngat ket noi voi server
```

```
db.close()
```

Khi chạy script này, nó sẽ cho kết quả sau trên thiết bị Linux.

```
Database version : 5.0.45
```

Nếu một kết nối được thành lập, thì một đối tượng Connection được trả về và được lưu giữ vào trong **db**, nếu không db được thiết lập là None. Tiếp đó, đối tượng **db** được sử dụng để tạo đối tượng **cursor**, mà tiếp đó được sử dụng để thực thi các truy vấn SQL. Cuối cùng, trước khi thoát ra, nó bảo đảm rằng kết nối tới Database được đóng và các resource được giải phóng.

Tạo bảng dữ liệu trong Python

Khi một kết nối tới Database đã được thành lập, chúng ta có thể tạo các bảng hoặc bản ghi vào trong bảng đó bởi sử dụng phương thức **execute** của đối tượng cursor đã được tạo.

Bạn theo dõi ví dụ để tạo bảng SINHVIEN:

```
import MySQLdb

# mo ket noi toi Database
db = MySQLdb.connect("localhost","testuser","test123","TESTDB" )

# chuan bi mot doi tuong cursor boi su dung phuong thuc cursor()
cursor = db.cursor()

# Xoa bang neu no da ton tai boi su dung phuong thuc execute().
cursor.execute("DROP TABLE IF EXISTS SINHVIEN")

# Tao mot bang
sql = """CREATE TABLE SINHVIEN (
    HO CHAR(20) NOT NULL,
    TEN CHAR(20),
    TUOI INT,
    GIOITINH CHAR(1),
```

```
HOCPHI FLOAT )"""\n\ncursor.execute(sql)\n\n# ngat ket noi voi server\ndb.close()
```

Hoạt động INSERT trong Python

Đây là hoạt động bắt buộc khi bạn muốn tạo các bản ghi vào trong bảng đã tạo.

Ví dụ sau sẽ thực thi lệnh SQL INSERT để tạo một bản ghi vào trong bảng SINHVIEN.

```
import MySQLdb\n\n# mo ket noi toi Database\ndb = MySQLdb.connect("localhost","testuser","test123","TESTDB" )\n\n# chuan bi mot doi tuong cursor boi su dung phuong thuc cursor()\ncursor = db.cursor()\n\n# Truy van SQL de INSERT mot ban ghi vao trong database.\nsql = """INSERT INTO SINHVIEN(HO,\n        TEN, TUOI, GIOITINH, HOCPHI)\n        VALUES ('Nguyen', 'Hoang', 20, 'M', 4000000)"""\ntry:\n    # Thuc thi lenh SQL\n    cursor.execute(sql)\n    # Commit cac thay doi vao trong Database\n    db.commit()\nexcept:\n    # Rollback trong tinh huong co bat ky error nao\n    db.rollback()
```

```
# ngat ket noi voi server  
db.close()
```

Ví dụ trên có thể được viết như sau để tạo các truy vấn SQL hay hơn:

```
import MySQLdb  
  
# mo ket noi toi Database  
db = MySQLdb.connect("localhost","testuser","test123","TESTDB" )  
  
# chuan bi mot doi tuong cursor boi su dung phuong thuc cursor()  
cursor = db.cursor()  
  
# Truy van SQL de INSERT mot ban ghi vao trong database.  
sql = "INSERT INTO SINHVIEN(HO, \  
          TEN, TUOI, GIOITINH, HOCPHI) \  
          VALUES ('%s', '%s', '%d', '%c', '%d' )" % \  
          ('Nguyen', 'Hoang', 20, 'M', 4000000)  
  
try:  
    # Thuc thi lenh SQL  
    cursor.execute(sql)  
    # Commit cac thay doi vao trong Database  
    db.commit()  
  
except:  
    # Rollback trong tinh huong co bat ky error nao  
    db.rollback()  
  
# ngat ket noi voi server  
db.close()
```

Ví dụ

Đoạn code sau là form thực thi khác, tại đây bạn có thể truyền các tham số một cách trực tiếp.

```
.....  
user_id = "test123"  
password = "password"  
  
con.execute('chen cac gia tri de dang nhap ("%s", "%s")' % \  
           (user_id, password))  
.....
```

Hoạt động đọc trong Python

Hoạt động đọc trên bất cứ Database nào nghĩa là lấy một số thông tin hữu ích từ Database.

Khi kết nối với Database được thiết lập, bạn có thể tạo một truy vấn vào trong Database này. Bạn có thể sử dụng hoặc phương thức **fetchone()** để lấy bản ghi đơn hoặc phương thức **fetchall()** để lấy nhiều giá trị từ một bảng.

- Hàm **fetchone()** lấy hàng tiếp theo của một tập kết quả truy vấn. Một tập kết quả là một đối tượng được trả về khi một đối tượng cursor được sử dụng để truy vấn một bảng.
- Hàm **fetchall()** lấy tất cả các hàng trong một tập kết quả. Nếu một số hàng đã sẵn sàng được trích từ tập kết quả đó, thì nó thu nhận các hàng còn lại từ tập kết quả.
- Thuộc tính **rowcount** là một thuộc tính read-only và trả về số hàng đã bị ảnh hưởng bởi phương thức **execute()**.

Ví dụ sau truy vấn tất cả bản ghi từ bảng SINHVIEN mà có salary lớn hơn 1000:

```
import MySQLdb  
  
# mo ket noi toi Database  
db = MySQLdb.connect("localhost", "testuser", "test123", "TESTDB" )  
  
# chuan bi mot doi tuong cursor boi su dung phuong thuc cursor()
```

```
cursor = db.cursor()

# Chuan bi truy van SQL de INSERT mot ban ghi vao trong database.
sql = "SELECT * FROM SINHVIEN \
       WHERE HOCPHI > '%d'" % (1000)

try:
    # Thuc thi lenh SQL
    cursor.execute(sql)

    # Lay tat ca cac hang trong list.
    results = cursor.fetchall()

    for row in results:
        ho = row[0]
        ten = row[1]
        tuoi = row[2]
        gioitinh = row[3]
        hocphi = row[4]

        # Bay gio in ket qua
        print "ho=%s,ten=%s,tuoi=%d,gioitinh=%s,hocphi=%d" % \
              (ho, ten, tuoi, gioitinh, hocphi )

except:
    print "Error: khong lay duoc du lieu"

# ngat ket noi voi server
db.close()
```

Kết quả là:

```
ho=Nguyen, ten=Hoang, tuoi=20, gioitinh=M, hocphi=4000000
```

Hoạt động Update trong Python

Hoạt động UPDATE trên bát cú Database nào nghĩa là để cập nhật một hoặc nhiều bản ghi mà đã có sẵn trong Database.

Thủ tục sau cập nhật tất cả bản ghi có GIOITINH là M. Ở đây, chúng ta tăng tất cả TUOI của male thêm 1 năm.

Ví dụ

```
import MySQLdb

# mo ket noi toi Database
db = MySQLdb.connect("localhost","testuser","test123","TESTDB" )

# chuan bi mot doi tuong cursor boi su dung phuong thuc cursor()
cursor = db.cursor()

# Truy van SQL de UPDATE cac ban ghi
sql = "UPDATE SINHVIEN SET TUOI = TUOI + 1
          WHERE GIOITINH = '%c'" % ('M')

try:
    # Thuc thi lenh SQL
    cursor.execute(sql)
    # Commit cac thay doi vao trong Database
    db.commit()
except:
    # Rollback trong tinh huong co bat ky error nao
    db.rollback()

# ngat ket noi voi server
db.close()
```

Hoạt động DELETE trong Python

Hoạt động DELETE là cần thiết khi bạn muốn xóa một số bản ghi từ Database. Dưới đây là thủ tục để xóa tất cả bản ghi từ bảng SINHVIEN với điều kiện là TUOI lớn hơn 20:

Ví dụ

```
import MySQLdb

# mo ket noi toi Database
db = MySQLdb.connect("localhost","testuser","test123","TESTDB" )

# chuan bi mot doi tuong cursor boi su dung phuong thuc cursor()
cursor = db.cursor()

# Chuan bi truy van SQL de DELETE cac ban ghi can thiet
sql = "DELETE FROM SINHVIEN WHERE TUOI > '%d'" % (20)

try:
    # Thuc thi lenh SQL
    cursor.execute(sql)
    # Commit cac thay doi vao trong Database
    db.commit()
except:
    # Rollback trong tinh huong co bat ky error nao
    db.rollback()

# ngat ket noi voi server
db.close()
```

Ngắt kết nối tới Database trong Python

Để ngắt kết nối tới Database, bạn sử dụng phương thức close(), có cú pháp như sau:

```
db.close()
```

Lập trình mạng trong Python

Chương này sẽ trình bày cho bạn hiểu về khái niệm quan trọng nhất trong lập trình mạng, đó là Lập trình Socket.

Socket là gì?

Socket là các điểm đầu nút (endpoint) của một kênh giao tiếp song hướng. Các Socket có thể giao tiếp bên trong một tiến trình, giữa các tiến trình trên cùng một thiết bị hoặc giữa các tiến trình trên các lục địa khác nhau.

Các Socket có thể được triển khai thông qua các kênh khác nhau: domain, TCP, UDP, ... Thư viện **socket** cung cấp các lớp riêng để xử lý các trình truyền tải cũng như một Interface chung để xử lý phần còn lại.

Socket có các khái niệm riêng như sau:

Khái niệm	Miêu tả
domain	Là family của các giao thức protocol được sử dụng như là kỹ thuật truyền tải. Các giá trị này là các hằng như AF_INET, PF_INET, PF_UNIX, PF_X25, ...
type	Kiểu giao tiếp giữa hai endpoint, đặc trưng là SOCK_STREAM cho các giao thức hướng kết nối (connection-oriented) và SOCK_DGRAM cho các giao thức không hướng kết nối
protocol	Đặc trưng là 0, mà có thể được sử dụng để nhận diện một biến thể của một giao thức bên trong một domain hoặc type
hostname	Định danh của một network interface: <ul style="list-style-type: none">• Một chuỗi, có thể là tên một host, địa chỉ IPV6, ...• Một chuỗi "<broadcast>", xác định một địa chỉ INADDR_BROADCAST• Một chuỗi có độ dài là 0, xác định INADDR_ANY, hoặc

	<ul style="list-style-type: none">Một số nguyên, được thông dịch dưới dạng một địa chỉ nhị phân trong thứ tự host byte
port	Mỗi Server nghe các lời gọi từ Client trên một hoặc nhiều cổng (port). Một port có thể là một chuỗi chứa số hiệu của port, một tên của một dịch vụ, ...

socket Module trong Python

Để tạo một Socket, bạn phải sử dụng hàm **socket.socket()** có sẵn trong socket Module, có cú pháp chung như sau:

```
s = socket.socket (socket_family, socket_type, protocol=0)
```

Chi tiết về tham số:

- socket_family:** Đây hoặc là AF_UNIX hoặc AF_INET.
- socket_type:** Đây hoặc là SOCK_STREAM hoặc SOCK_DGRAM.
- protocol:** Thường được để trống, mặc định là 0.

Khi bạn đã có đối tượng socket, bạn có thể sử dụng các hàm để tạo chương trình cho Client hoặc Server. Dưới đây là danh sách các hàm:

Các phương thức sử dụng cho Server Socket

Phương thức	Miêu tả
s.bind()	Phương thức này gắn kết địa chỉ (hostname, port number) tới Socket
s.listen()	Phương thức này thiết lập và bắt đầu TCP Listener.
s.accept()	Phương thức này chấp nhận một cách thụ động kết nối TCP Client, đợi cho tới khi kết nối tới.

Các phương thức sử dụng cho Client Socket

s.connect(): Phương thức này khởi tạo kết nối TCP Server.

Các phương thức chung cho Socket

Phương thức	Miêu tả
s.recv()	Phương thức này nhận TCP message.
s.send()	Phương thức này truyền TCP message.
s.recvfrom()	Phương thức này nhận UDP message.
s.sendto()	Phương thức này truyền UDP message.
s.close()	Phương thức này đóng Socket.
socket.gethostname()	Trả về hostname.

Ví dụ viết một chương Server đơn giản trong Python

Để viết một Server, bạn sử dụng hàm **socket** có trong socket Module để tạo một đối tượng socket. Sau đó, đối tượng socket được sử dụng để gọi các hàm khác để thiết lập một Socket Server.

Bây giờ gọi hàm **bind(hostname, port)** để xác định một port cho dịch vụ của bạn trên host đã cho.

Tiếp đó, gọi phương thức **accept** của đối tượng được trả về. Phương thức này đợi tới khi một Client kết nối tới port mà bạn đã xác định, và sau đó trả về một đối tượng **connection** mà biểu diễn kết nối tới Client đó.

```
# Day la server.py file

import socket          # Import socket module

s = socket.socket()      # Tao mot doi tuong socket
host = socket.gethostname() # Lay ten thiet bi local
port = 12345            # Danh rieng mot port cho dich vu cua ban.
s.bind((host, port))     # Ket noi toi port
```

```
s.listen(5)          # Doi 5 s de ket noi voi client.

while True:
    c, addr = s.accept()      # Thiet lap ket noi voi client.
    print 'Da ket noi voi', addr
    c.send('Cam on ban da ket noi')
    c.close()                 # Ngat ket noi
```

Ví dụ viết một chương trình Client đơn giản trong Python

Chúng ta viết một chương trình Client đơn giản để mở một kết nối tới một port có số hiệu đã cho là 12345 và với host đã xác định.

Hàm **socket.connect(hostname, port)** mở một kết nối TCP tới **hostname** trên **port** đã cho. Khi bạn có một socket đã được mở, bạn có thể đọc từ nó giống như bất kỳ đối tượng IO nào.

Code sau là một Client rất đơn giản để kết nối tới host và port đã cho, đọc bất cứ dữ liệu nào có sẵn từ Socket đó và sau đó thoát.

```
# This is client.py file

import socket           # Import socket module

s = socket.socket()      # Tao mot doi tuong socket
host = socket.gethostname() # Lay ten thiet bi local
port = 12345              # Danh rieng mot port cho dich vu cua ban.

s.connect((host, port))
print s.recv(1024)
s.close                  # Dong socket
```

Bây giờ chạy server.py trong background và sau đó chạy client.py trên để xem kết quả.

```
# Chay server trong background.
$ python server.py &
```

```
# Mot khi server da bat dau, ban chay client nhu sau:  
  
$ python client.py
```

Kết quả là:

```
Da ket noi voi ('127.0.0.1', 48437)  
Cam on ban da ket noi
```

Các Module quan trọng trong lập trình mạng

Protocol	Tính năng chung	Port No	Python module
HTTP	Web page	80	httplib, urllib, xmlrpclib
NNTP	Usenet new	119	nntplib
FTP	Truyền tải file	20	ftplib, urllib
SMTP	Gửi email	25	smtplib
POP3	Lấy email	110	poplib
IMAP4	Lấy email	143	imaplib
Telnet	Command line	23	telnetlib
Gopher	Truyền tải Document	70	gopherlib, urllib

Gửi Email sử dụng SMTP trong Python

SMTP (là viết tắt của Simple Mail Transfer Protocol) là một giao thức để xử lý trình gửi và định tuyến email giữa các Mail Server. Python cung cấp **smtplib** Module, mà định nghĩa một đối tượng SMTP Client Session có thể được sử dụng để gửi email tới bất kỳ thiết bị internet nào với một SMTP hoặc ESMTP Listener.

Dưới đây là cú pháp cơ bản để tạo một đối tượng SMTP:

```
import smtplib

smtpObj = smtplib.SMTP( [host [, port [, local_hostname]]] )
```

Chi tiết về tham số:

- **host:** Đây là host đang chạy SMTP Server của bạn. Bạn có thể xác định địa chỉ IP của host hoặc một tên miền như vietjack.com. Đây là tham số tùy ý.
- **port:** Nếu bạn cung cấp tham số host, thì bạn cần xác định một port, đây là nơi SMTP Server nghe yêu cầu. Thường thì port này sẽ là 25.
- **ten_localhost:** Nếu SMTP Server của bạn đang chạy trên thiết bị local, thì bạn có thể xác định là **localhost** cho tùy chọn này.

Đối tượng SMTP có một phương thức instance là **sendmail**, được sử dụng để gửi một thông điệp. Nó nhận ba tham số:

- *sender* - Là một chuỗi chỉ địa chỉ của người gửi.
- *receivers* - Một danh sách các chuỗi, mỗi chuỗi là địa chỉ của người nhận.
- *message* - Là một thông điệp dưới định dạng chuỗi.

Ví dụ

Dưới đây là cách đơn giản để gửi một email bởi sử dụng Python.

```
import smtplib
```

```
sender = 'from@fromdomain.com'

receivers = ['to@todomain.com']

message = """From: Tu <from@fromdomain.com>
To: Toi <to@todomain.com>
Subject: SMTP e-mail test

Day la vi du ve gui email.

"""

try:
    smtpObj = smtplib.SMTP('localhost')
    smtpObj.sendmail(sender, receivers, message)
    print "Gui email thanh cong"
except SMTPException:
    print "Error: khong the gui email"
```

Trong ví dụ trên, chúng ta đã đặt một email cơ bản trong message, bởi sử dụng trích dẫn tam. Bạn cần định dạng các trường header một cách chính xác. Một email cần một **From**, **To**, và **Subject** header, được phân biệt với phần thân email bởi một dòng trắng.

Để gửi một email, bạn sử dụng **smtpObj** để kết nối tới SMTP Server trên thiết bị local và sau đó sử dụng phương thức **sendmail** cùng với thông điệp message, địa chỉ người gửi, địa chỉ người nhận là các tham số.

Nếu bạn không chạy SMTP Server trên thiết bị local, bạn có thể sử dụng **smtplib** Client để giao tiếp với một SMTP Server từ xa. Ví dụ:

```
smtplib.SMTP('mail.your-domain.com', 25)
```

Gửi HTML email bởi sử dụng Python

Khi bạn gửi một text message bởi sử dụng Python, thì tất cả nội dung được xem như dưới dạng text đơn giản. Ngay cả khi bạn bao các HTML tag trong thông điệp text này, thì nó cũng chỉ hiển thị dưới dạng text đơn giản và các thẻ HTML này sẽ không được định dạng tương ứng với cú pháp

HTML. Nhưng Python cung cấp tùy chọn để gửi một HTML message dưới dạng một thông điệp HTML thực sự.

Trong khi gửi một email message, bạn có thể xác định một Mine version, kiểu nội dung, và bộ ký tự để gửi một HTML email.

Ví dụ

Ví dụ đơn giản này gửi một HTML content dưới dạng một email.

```
import smtplib

message = """From: Tu <from@fromdomain.com>
To: Toi <to@todomain.com>
MIME-Version: 1.0
Content-type: text/html
Subject: SMTP HTML e-mail test

Day la vi du gui email trong dinh dang HTML

<b>Day la HTML message.</b>
<h1>Day la headline.</h1>
"""

try:
    smtpObj = smtplib.SMTP('localhost')
    smtpObj.sendmail(sender, receivers, message)
    print "Gui email thanh cong"
except SMTPException:
    print "Error: khong the gui email"
```

Gửi đính kèm với email bởi sử dụng Python

Để làm điều này, bạn cần thiết lập trường Content-type header thành **multipart/mixed**. Sau đó, các phần text và attachment có thể được xác định bên trong **boundaries**.

Một boundary được bắt đầu với hai dấu gạch nối (--) được sau bởi một số duy nhất, mà không thể xuất hiện trong phần thông điệp của email. Một final boundary, biểu thị khu vực cuối cùng của email đó, cũng phải kết thúc với hai dấu gạch dưới.

Các file đính kèm nên được bao quanh bởi hàm pack("m") để có mã hóa base64 trước khi truyền tải.

Ví dụ

Ví dụ sau sẽ gửi một file có tên là /tmp/test.txt dưới dạng một đính kèm.

```
import smtplib  
  
import base64  
  
filename = "/tmp/test.txt"  
  
# Doc mot file va ma hoa no trong dinh dang base64  
fo = open(filename, "rb")  
filecontent = fo.read()  
encodedcontent = base64.b64encode(filecontent) # base64  
  
sender = 'webmaster@vietjack.com'  
reciever = 'amrood.admin@gmail.com'  
  
marker = "AUNIQUEMARKER"  
  
body = """  
Day la vi du de gui email voi attachment.  
"""  
# Dinh nghia cac header.  
part1 = """From: Tu <me@fromdomain.net>  
To: Toi <amrood.admin@gmail.com>  
Subject: Sending Attachement
```

```
MIME-Version: 1.0

Content-Type: multipart/mixed; boundary=%s

--%s

""" % (marker, marker)

# Dinh nghia message action

part2 = """Content-Type: text/plain

Content-Transfer-Encoding:8bit


%s

--%s

""" % (body,marker)

# Dinh nghia phan attachment

part3 = """Content-Type: multipart/mixed; name=\"%s\"

Content-Transfer-Encoding:base64

Content-Disposition: attachment; filename=%s


%s

--%s--

""" %(filename, filename, encodedcontent, marker)

message = part1 + part2 + part3

try:

    smtpObj = smtplib.SMTP('localhost')

    smtpObj.sendmail(sender, reciever, message)

    print "Gui email thanh cong"

except Exception:

    print "Error: khong the gui email"
```

Đa luồng (Multithread) trong Python

Một chương trình đa luồng chứa hai hoặc nhiều phần mà có thể chạy đồng thời và mỗi phần có thể xử lý tác vụ khác nhau tại cùng một thời điểm, để sử dụng tốt nhất các nguồn có sẵn, đặc biệt khi máy tính của bạn có nhiều CPU.

Python cung cấp thread Module và threading Module để bạn có thể bắt đầu một thread mới cũng như một số tác vụ khác trong khi lập trình đa luồng. Mỗi một Thread đều có vòng đời chung là bắt đầu, chạy và kết thúc. Một Thread có thể bị ngắt (interrupt), hoặc tạm thời bị dừng (sleeping) trong khi các Thread khác đang chạy – được gọi là yielding.

Bắt đầu một Thread mới trong Python

Phương thức dưới đây có sẵn trong thread Module được sử dụng để bắt đầu một Thread mới:

```
thread.start_new_thread ( function, args[, kwargs] )
```

Lời gọi phương thức này được trả về ngay lập tức và Thread con bắt đầu và gọi hàm **function** với danh sách các tham số args đã truyền. Khi hàm function trả về, thì Thread kết thúc.

Ở đây, **args** là một Tuple của các tham số, sử dụng một Tuple trống để gọi hàm function mà không truyền cho nó bất kỳ tham số nào. Tham số **kwargs** là một Dictionary của các tham số từ khóa tùy ý. (Bạn thao khảo chương Hàm trong Python để biết chi tiết tham số từ khóa là gì)

Ví dụ

```
import thread
import time

# Định nghĩa một hàm cho thread
def print_time( threadName, delay):
    count = 0
    while count < 5:
        time.sleep(delay)
        count += 1
        print "%s: %s" % ( threadName, time.ctime(time.time()) )
```

```
# Tao hai thread nhu sau
try:
    thread.start_new_thread( print_time, ("Thread-1", 2, ) )
    thread.start_new_thread( print_time, ("Thread-2", 4, ) )
except:
    print "Error: khong the bat dau thread"

while 1:
    pass
```

Kết quả là:

```
Thread-1: Mon Nov 21 15:42:17 2015
Thread-1: Mon Nov 21 15:42:19 2015
Thread-2: Mon Nov 21 15:42:19 2015
Thread-1: Mon Nov 21 15:42:21 2015
Thread-2: Mon Nov 21 15:42:23 2015
Thread-1: Mon Nov 21 15:42:23 2015
Thread-1: Mon Nov 21 15:42:25 2015
Thread-2: Mon Nov 21 15:42:27 2015
Thread-2: Mon Nov 21 15:42:31 2015
Thread-2: Mon Nov 21 15:42:35 2015
```

Mặc dù thread Module rất hiệu quả với đa luồng tầm thấp nhưng khi so sánh với **threadingModule** thì nó có nhiều điểm hạn chế. Phần tiếp theo giới thiệu về threading Module.

threading Module trong Python

Module mới này được bao với Python 2.4 nhằm cung cấp nhiều hỗ trợ mạnh mẽ và cấp độ cao hơn cho các Thread trong khi so sánh với thread Module ở trên. Ngoài các phương thức có trong thread Module, thì threading Module còn bổ sung thêm một số phương thức khác, đó là:

- **threading.activeCount()**: Trả về số đối tượng thread mà là active.
- **threading.currentThread()**: Trả về số đối tượng thread trong Thread control của Caller.

- **threading.enumerate()**: Trả về một danh sách tất cả đối tượng thread mà hiện tại là active.

Bên cạnh đó, threading Module có lớp Thread để triển khai đa luồng. Lớp này có các phương thức sau:

- **run()**: Là entry point cho một Thread.
- **start()**: Bắt đầu một thread bởi gọi phương thức run().
- **join([time])**: Đợi cho các thread kết thúc.
- **isAlive()**: Kiểm tra xem một thread có đang thực thi hay không.
- **getName()**: Trả về tên của một thread.
- **setName()**: Thiết lập tên của một thread.

Tạo Thread bởi sử dụng threading Module trong Python

Để triển khai một thread mới bởi sử dụng threading Module, bạn phải thực hiện:

- Định nghĩa một lớp con của lớp *Thread*.
- Ghi đè phương thức `__init__(self [,args])` để bổ sung thêm các tham số.
- Sau đó, ghi đè phương thức `run(self [,args])` để triển khai những gì thread cần thực hiện khi được bắt đầu.

Một khi bạn đã tạo lớp con Thread mới, bạn có thể tạo một instance của nó và sau đó bắt đầu một Thread bởi triệu hồi phương thức **start()**.

Ví dụ

```
import threading  
import time  
  
exitFlag = 0
```

```
class myThread (threading.Thread):  
    def __init__(self, threadID, name, counter):  
        threading.Thread.__init__(self)  
        self.threadID = threadID  
        self.name = name  
        self.counter = counter  
  
    def run(self):  
        print "Bat dau " + self.name  
        print_time(self.name, self.counter, 5)  
        print "Ket thuc " + self.name  
  
  
def print_time(threadName, delay, counter):  
    while counter:  
        if exitFlag:  
            threadName.exit()  
        time.sleep(delay)  
        print "%s: %s" % (threadName, time.ctime(time.time()))  
        counter -= 1  
  
  
# Tao cac thread moi  
thread1 = myThread(1, "Thread-1", 1)  
thread2 = myThread(2, "Thread-2", 2)  
  
  
# Bat dau cac thread moi  
thread1.start()  
thread2.start()  
  
  
print "Ket thuc Main Thread"
```

Kết quả là:

```
Bat dau Thread-1
```

```
Bat dau Thread-2
```

```
Ket thuc Main Thread
```

```
Thread-1: Mon Nov 21 09:10:03 2015
Thread-1: Mon Nov 21 09:10:04 2015
Thread-2: Mon Nov 21 09:10:04 2015
Thread-1: Mon Nov 21 09:10:05 2015
Thread-1: Mon Nov 21 09:10:06 2015
Thread-2: Mon Nov 21 09:10:06 2015
Thread-1: Mon Nov 21 09:10:07 2015
Ket thuc Thread-1
Thread-2: Mon Nov 21 09:10:08 2015
Thread-2: Mon Nov 21 09:10:10 2015
Thread-2: Mon Nov 21 09:10:12 2015
Ket thuc Thread-2
```

Đồng bộ hóa các Thread trong Python

Python cung cấp threading Module, mà bao gồm một kỹ thuật locking cho phép bạn đồng bộ hóa các Thread một cách dễ dàng. Một lock mới được tạo bởi gọi phương thức Lock().

Phương thức **acquire(blocking)** của đối tượng lock mới này được sử dụng để ép các Thread chạy một cách đồng bộ. Tham số *blocking* tùy ý cho bạn khả năng điều khiển để xem một Thread có cần đợi để đạt được lock hay không.

Nếu tham số *blocking* được thiết lập là 0, tức là Thread ngay lập tức trả về một giá trị 0 nếu không thu được lock và trả về giá trị 1 nếu thu được lock. Nếu *blocking* được thiết lập là 1, thì Thread cần đợi cho đến khi lock được giải phóng.

Phương thức **release()** của đối tượng lock được sử dụng để giải phóng lock khi nó không cần nữa.

Ví dụ

```
import threading
import time

class myThread (threading.Thread):
    def __init__(self, threadID, name, counter):
```

```
threading.Thread.__init__(self)
self.threadID = threadID
self.name = name
self.counter = counter

def run(self):
    print "Bat dau " + self.name
    # Lay lock de dong bo hoa cac thread
    threadLock.acquire()
    print_time(self.name, self.counter, 3)
    # Giai phong lock cho thread ke tiep
    threadLock.release()

def print_time(threadName, delay, counter):
    while counter:
        time.sleep(delay)
        print "%s: %s" % (threadName, time.ctime(time.time()))
        counter -= 1

threadLock = threading.Lock()
threads = []

# Tao cac thread moi
thread1 = myThread(1, "Thread-1", 1)
thread2 = myThread(2, "Thread-2", 2)

# Bat dau cac thread moi
thread1.start()
thread2.start()

# Them cac thread vao list
threads.append(thread1)
threads.append(thread2)
```

```
# Doi cho tat ca thread ket thuc  
  
for t in threads:  
    t.join()  
  
print "Ket thuc Main Thread"
```

Kết quả là:

```
Bat dau Thread-1  
  
Bat dau Thread-2  
  
Thread-1: Mon Nov 21 09:11:28 2015  
  
Thread-1: Mon Nov 21 09:11:29 2015  
  
Thread-1: Mon Nov 21 09:11:30 2015  
  
Thread-2: Mon Nov 21 09:11:32 2015  
  
Thread-2: Mon Nov 21 09:11:34 2015  
  
Thread-2: Mon Nov 21 09:11:36 2015  
  
Ket thuc Main Thread
```

Queue Module: quyền ưu tiên đa luồng trong Python

Queue Module cho phép bạn tạo một đối tượng queue mới mà có thể giữ một số lượng item nào đó. Dưới đây là các phương thức:

- **get()**: Xóa và trả về một item từ queue.
- **put()**: Thêm một item tới một queue.
- **qsize()** : Trả về số item mà hiện tại đang trong queue.
- **empty()**: Trả về true nếu queue là trống, nếu không thì trả về false.
- **full()**: Trả về true nếu queue là đầy, nếu không thì trả về false.

Ví dụ

```
import Queue  
  
import threading  
  
import time
```

```
exitFlag = 0

class myThread (threading.Thread):
    def __init__(self, threadID, name, q):
        threading.Thread.__init__(self)
        self.threadID = threadID
        self.name = name
        self.q = q
    def run(self):
        print "Bat dau " + self.name
        process_data(self.name, self.q)
        print "Ket thuc " + self.name

def process_data(threadName, q):
    while not exitFlag:
        queueLock.acquire()
        if not workQueue.empty():
            data = q.get()
            queueLock.release()
            print "%s dang xu ly %s" % (threadName, data)
        else:
            queueLock.release()
            time.sleep(1)

threadList = ["Thread-1", "Thread-2", "Thread-3"]
nameList = ["One", "Two", "Three", "Four", "Five"]
queueLock = threading.Lock()
workQueue = Queue.Queue(10)
threads = []
threadID = 1

# Tao cac thread moi
```

```
for tName in threadList:  
    thread = myThread(threadID, tName, workQueue)  
    thread.start()  
    threads.append(thread)  
    threadID += 1  
  
# Dien vao queue  
queueLock.acquire()  
for word in nameList:  
    workQueue.put(word)  
queueLock.release()  
  
# Doi den khi queue la trong  
while not workQueue.empty():  
    pass  
  
# Thong bao cho thread do la thoi gian de ket thuc  
exitFlag = 1  
  
# Doi cho tat ca thread ket thuc  
for t in threads:  
    t.join()  
print "Ket thuc Main Thread"
```

Kết quả là:

```
Bat dau Thread-1  
Bat dau Thread-2  
Bat dau Thread-3  
Thread-1 dang xu ly One  
Thread-2 dang xu ly Two  
Thread-3 dang xu ly Three  
Thread-1 dang xu ly Four  
Thread-2 dang xu ly Five  
Ket thuc Thread-3
```

```
Ket thuc Thread-1
```

```
Ket thuc Thread-2
```

```
Ket thuc Main Thread
```

Xử lý XML trong Python

Thư viện Python chuẩn cung cấp các Interface hữu ích để làm việc với XML. Hai APIs cơ bản và được sử dụng nhiều nhất là SAX và DOM. SAX (viết tắt của Simple API for XML) là read-only trong khi DOM (viết tắt của Document Object Model) cho phép tạo các thay đổi tới XML file.

Chương này sẽ giới thiệu về cả hai Interface này, nhưng trước hết, chúng ta tạo một XML file đơn giản có tên là movies.xml để làm input:

Phân tích cú pháp XML với SAX APIs

Nói chung, bạn cần tạo riêng cho mình một ContentHandler là lớp con của `xml.sax.ContentHandler`.

ContentHandler của bạn sẽ xử lý các tag cụ thể và các thuộc tính của XML. Một đối tượng ContentHandler cung cấp các phương thức để xử lý các sự kiện parsing khác nhau.

Phương thức `startDocument` và `endDocument` được gọi tại phần bắt đầu và phần cuối của XML file. Phương thức `characters(text)` để truyền dữ liệu ký tự của XML thông qua tham số `text`.

Đối tượng ContentHandler được gọi tại phần bắt đầu và phần cuối của mỗi phần tử. Nếu Parser không trong namespace mode, thì các phương thức `startElement(tag, thuoc_tinh)` và `endElement(tag)` được gọi; nếu không thì, các phương thức tương ứng `startElementNS` và `endElementNS` được gọi. Ở đây, tham số `tag` là thẻ và `thuoc_tinh` là một đối tượng `Attributes`.

Bạn tìm hiểu một số phương thức quan trọng sau để hiểu rõ tiến trình xử lý hơn:

Phương thức make_parser trong Python

Phương thức sau tạo một đối tượng parser mới và trả về nó. Đối tượng parser đã được tạo này sẽ là kiểu parser đầu tiên mà hệ thống tìm thấy.

```
xml.sax.make_parser( [parser_list] )
```

Tham số `parser_list` là tùy ý, bao gồm một danh sách các parser để sử dụng, tất cả phải triển khai phương thức `make_parser`.

Phương thức parse trong Python

Phương thức này tạo một SAX parser và sử dụng nó để phân tích cú pháp một tài liệu.

```
xml.sax.parse( xmlfile, contenthandler[, errorhandler] )
```

Chi tiết tham số:

- **xmlfile:** Đây là tên của XML file để đọc từ đó.
- **contenthandler:** Đây phải là một đối tượng ContentHandler.
- **errorhandler:** Nếu được xác định, thì nó phải là một đối tượng SAX ErrorHandler.

Phương thức parseString trong Python

Phương thức này cũng dùng để tạo một SAX parser và để phân tích cú pháp *XML string* đã cho.

```
xml.sax.parseString(xmlstring, contenthandler[, errorhandler])
```

Chi tiết về tham số:

- **xmlstring:** Là tên của XML string để đọc từ đó.
- **contenthandler:** Phải là một đối tượng ContentHandler.
- **errorhandler:** Nếu được xác định, thì nó phải là một đối tượng SAX ErrorHandler.

Ví dụ

```
import xml.sax

class Phim_BoHandler( xml.sax.ContentHandler ):

    def __init__(self):
        self.CurrentData = ""
        self.type = ""
        self.format = ""
        self.year = ""
        self.rating = ""
        self.stars = ""
        self.description = ""

    # Call when an element starts
    def startElement(self, tag, attrs):
```

```
# Goi khi mot phan tu bat dau
def startElement(self, tag, attributes):
    self.CurrentData = tag
    if tag == "movie":
        print "*****Phim Bo*****"
        title = attributes["title"]
        print "Ten Phim:", title

# Goi khi mot phan tu ket thuc
def endElement(self, tag):
    if self.CurrentData == "type":
        print "The loai:", self.type
    elif self.CurrentData == "format":
        print "Dinh dang:", self.format
    elif self.CurrentData == "year":
        print "Nam:", self.year
    elif self.CurrentData == "rating":
        print "Rating:", self.rating
    elif self.CurrentData == "stars":
        print "Dien vien:", self.stars
    elif self.CurrentData == "description":
        print "Gioi thieu:", self.description
    self.CurrentData = ""

# Goi khi mot ky tu duoc doc
def characters(self, content):
    if self.CurrentData == "type":
        self.type = content
    elif self.CurrentData == "format":
        self.format = content
    elif self.CurrentData == "year":
        self.year = content
    elif self.CurrentData == "rating":
```

```
    self.rating = content
    elif self.CurrentData == "stars":
        self.stars = content
    elif self.CurrentData == "description":
        self.description = content

if ( __name__ == "__main__"):

    # Tao mot XMLReader
    parser = xml.sax.make_parser()
    # Tat cac namespace
    parser.setFeature(xml.sax.handler.feature_namespaces, 0)

    # ghi de ContextHandler mac dinh
    Handler = Phim_BoHandler()
    parser.setContentHandler( Handler )

    parser.parse("movies.xml")
```

Phân tích cú pháp XML với DOM APIs

DOM thực sự hữu ích với các ứng dụng truy cập ngẫu nhiên. SAX chỉ cho phép bạn xem một bit của tài liệu tại một thời điểm và không có quyền truy cập khác.

Dưới đây là cách nhanh nhất để tải một XML document và để tạo một đối tượng minidom bởi sử dụng `xml.dom` Module. Đối tượng minidom cung cấp một phương thức `parser` đơn giản mà tạo một DOM tree một cách nhanh chóng từ XML file.

Hàm `parse(file [,parser])` của đối tượng minidom để phân tích cú pháp XML file đã được chỉ rõ bởi file bên trong một đối tượng DOM tree.

```
from xml.dom.minidom import parse
import xml.dom.minidom
```

```
# Mo mot tai lieu XML document boi su dung minidom parser
DOMTree = xml.dom.minidom.parse("movies.xml")
collection = DOMTree.documentElement
if collection.hasAttribute("shelf"):
    print "Root element : %s" % collection.getAttribute("shelf")

# Lay tat ca phim trong bo suu tap
movies = collection.getElementsByTagName("movie")

# in chi tiet ve moi phim.
for movie in movies:
    print "*****Phim Bo*****"
    if movie.hasAttribute("title"):
        print "Ten Phim: %s" % movie.getAttribute("title")

    type = movie.getElementsByTagName('type')[0]
    print "The loai: %s" % type.childNodes[0].data
    format = movie.getElementsByTagName('format')[0]
    print "Dinh dang: %s" % format.childNodes[0].data
    rating = movie.getElementsByTagName('rating')[0]
    print "Rating: %s" % rating.childNodes[0].data
    description = movie.getElementsByTagName('description')[0]
    print "Gioi thieu: %s" % description.childNodes[0].data
```

Tool/Utility trong Python

Chương này giới thiệu một số Module hữu ích mà bạn nên biết có trong thư viện Python chuẩn.

dis Module trong Python

dis Module là disassembler của Python. Nó chuyển đổi byte code thành một định dạng chính xác hơn cho người dùng. Bạn có thể chạy disassembler này từ command line. Nó thông dịch script đã cho và in byte code đã được tách rời STDOUT. Bạn cũng có thể sử dụng nó như là một Module. Hàm dis nhận một lớp, phương thức, hàm hoặc đối tượng code như là tham số đơn của nó. Ví dụ:

Ví dụ

```
import dis

def sum():
    vara = 10
    varb = 20

    sum = vara + varb
    print "vara + varb = %d" % sum

# Gọi ham dis.

dis.dis(sum)
```

Kết quả là:

6	0 LOAD_CONST	1 (10)
	3 STORE_FAST	0 (vara)
7	6 LOAD_CONST	2 (20)
	9 STORE_FAST	1 (varb)
9	12 LOAD_FAST	0 (vara)
	15 LOAD_FAST	1 (varb)

```
18 BINARY_ADD
19 STORE_FAST      2 (sum)

10      22 LOAD_CONST      3 ('vara + varb = %d')
25 LOAD_FAST      2 (sum)
28 BINARY_MODULO
29 PRINT_ITEM
30 PRINT_NEWLINE
31 LOAD_CONST      0 (None)
34 RETURN_VALUE
```

pdb Module trong Python

pdb Module là Debugger chuẩn của Python. Nó được dựa trên bdb Debugger Framework.

Bạn có thể chạy Debugger này từ command line (gõ n hoặc next để tới dòng tiếp theo và help để lấy danh sách các lệnh có sẵn).

Ví dụ

Trước khi bạn chạy pdb.py, thiết lập path của mình một cách thích hợp tới thư mục Python lib. Sau đó bạn có thể thử ví dụ sum.py trên.

```
$pdb.py sum.py
> /test/sum.py(3)<module>()
-> import dis
(Pdb) n
> /test/sum.py(5)<module>()
-> def sum():
(Pdb) n
>/test/sum.py(14)<module>()
-> dis.dis(sum)
(Pdb) n
   6      0 LOAD_CONST      1 (10)
         3 STORE_FAST      0 (vara)

   7      6 LOAD_CONST      2 (20)
```

```
 9 STORE_FAST           1 (varb)

 9          12 LOAD_FAST           0 (vara)
 15 LOAD_FAST           1 (varb)
 18 BINARY_ADD
 19 STORE_FAST           2 (sum)

10          22 LOAD_CONST         3 ('vara + varb = %d')
 25 LOAD_FAST           2 (sum)
 28 BINARY_MODULO
 29 PRINT_ITEM
 30 PRINT_NEWLINE
 31 LOAD_CONST           0 (None)
 34 RETURN_VALUE

--Return--
> /test/sum.py(14)<module>()->None
-v dis.dis(sum)
(Pdb) n
--Return--
> <string>(1)<module>()->None
(Pdb)
```

profile Module trong Python

profile Module là Profiler chuẩn của Python. Bạn có thể chạy nó từ command line.

Ví dụ

Chúng ta thử profile chương trình sau:

```
vara = 10
varb = 20

sum = vara + varb
```

```
print "vara + varb = %d" % sum
```

Bây giờ thử chạy **cProfile.py** thông qua *sum.py* như sau:

```
$cProfile.py sum.py
vara + varb = 30

    4 function calls in 0.000 CPU seconds

Ordered by: standard name

ncalls  tottime  percall  cumtime  percall filename:lineno
      1    0.000    0.000    0.000    0.000 <string>:1(<module>)
      1    0.000    0.000    0.000    0.000 sum.py:3(<module>)
      1    0.000    0.000    0.000    0.000 {execfile}
      1    0.000    0.000    0.000    0.000 {method .....}
```

tabnanny Module trong Python

tabnanny Module kiểm tra các source file của Python xem có độ th деят dòng nào mơ hồ hay không. Nếu một file mà xóa trộn tab và space, thì tất nhiên là không vấn đề gì với kích cỡ tab bạn đang sử dụng, nhưng tabnanny sẽ đưa ra lời phàn nàn như trong ví dụ sau:

Ví dụ

Chúng ta thử profile chương trình sau:

```
vara = 10
varb = 20

sum = vara + varb
print "vara + varb = %d" % sum
```

Nếu bạn thử với *tabnanny.py*, thì nó sẽ đưa ra lời phàn nàn:

```
$tabnanny.py -v sum.py
'sum.py': Clean bill of health.
```

