Lập trình Python cơ bản

pythonvietnam

Published with GitBook

Mục Lục

Giới thiệu chung

Chào các bạn,

Ngôn ngữ lập trình Python ngày càng được phổ biến và ứng dụng rộng rãi trong IT. Nắm bắt được xu hướng đó và nhằm tạo điều kiện nghiên cứu cho các sinh viên, người mới bắt đầu tiếp xúc với Python, cộng đồng Python Việt Nam sưu tầm và biên tập lại giới thiệu cài nhìn tổng quan nhất về Python.

Trong quá trình sưu tâm, biên dịch chắc chắn không thể tránh khỏi các thiếu sót, rất mong được sự đóng góp hơn nữa của các đồng nghiệp, anh em bạn bè để hoàn thiện hơn nữa.

Mọi ý kiến đóng góp xin gửi về:

Nguyễn Ngọc Khánh

Email: khanhnn@pythonvietnam.info

Skype/ Yahoo: khanhnnvn

http://pythonvietnam.info

http://pythonvietnam.com

Facebook: FanPage, Group

Cảm ơn sự quan tâm, ủng hộ của các bạn!

Nhóm dịch giả.

Cài đặt Python

Trong phần này, chúng ta sẽ nói về vấn đề cài đặt Python.

a. Cài đặt Python trên Windows:

Bước 1: Thực hiện tải bản cài đặt từ trang chủ: http://www.python.org. Chú ý phiên bản được chúng tôi sử dụng trong tài liệu này là Python 2.x vì vậy bạn nên lựa chọn phiên bản phù hợp cho việc thực hành.

Bước 2: Tiến hành cài đặt Python như một chương trình trên Windows bình thường.

Bước 3: Để có thể làm việc với Python ở cửa sổ cmd mà không cần trỏ rõ thư mục cài đặt, bạn cần thiết lập biến môi trường bằng cách:

```
Click chuột phải vào Mycomputer chọn Properties

Click chọn Advanced system settting.

Chọn Environment Variables

Sau đó thêm đường dẫn cài đặt Python vào biến PATH
```

Sau đó bạn gõ python tại của sổ cmd, nếu hiển thị như sau là bạn đã thành công.

```
Python 2.7.6 (default, Nov 10 2013, 19:24:18) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
```

b. Cài đặt trên HĐH Linux

Hầu như các hệ điều hành đều hỗ trợ Python, bạn có thể cài đặt online, hoặc offline. Chúng ta sẽ thấy các cách phổ biến sau:

Trên Debian (Ubuntu, Linux Mint, ...) chúng ta sử dụng

```
apt-get install python
```

Trên RedHat (CentOS, RedHat, Asianux, ...)

```
yum install python
```

Sau khi cài đặt xong, bạn cũng có thể sử dụng Python bằng cách gõ python vào cửa sổ dòng lệnh.

Chúc các bạn thành công !

Bắt đầu

Python là ngôn ngữ lập trình thông dịch, điều đó có nghĩa là bạn có thể viết code ngay trên trình thông dịch hoặc viết vào file sau đó chạy chúng. Đầu tiên chúng ta sẽ sử dụng trình thông dịch để bắt đầu viết một chương trình đầu tiên (bạn có thể sử dụng shell trên Windows hoặc Terminal trên Linux)

```
$ python
Python 2.5.1 (r251:54863, Oct 30 2007, 13:54:11)
q[GCC 4.1.2 20070925 (Red Hat 4.1.2-33)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Để có thể in ra dòng chữ "Xin chào Python Viet Nam" bạn có thể làm như sau:

```
>>> print "Xin chào Python Viet Nam!"
Hello World!

helloworld.py
```

Bây giờ ta sẽ làm cách mà lập trình viên hay làm, bạn sẽ tạo một file helloworld.py, bạn có thể sử dụng bất cứ một trình soạn thảo mà bạn biết để tạo ra file này. Chú ý đuôi mở rộng là .py

```
#!/usr/bin/env python
print "Hello World!"
```

Để thực hiện chạy, bạn cần trao quyền cho file đó được thực thi. Bạn dùng lệnh sau:

```
$ chmod +x helloworld.py
```

Sau đó chạy

```
$ ./helloworld.py
Hello World!
```

Trong dòng đầu tiên có dấu #!, chúng ta gọi nó là sha-bang. Sử dụng để thông báo cho trình thông dịch Python chạy đoạn code. Dòng tiếp theo trong đoạn code trên chúng ta in ra thông báo. Trong Python chúng ta gọi các dòng văn bản là chuỗi.

Khoảng trắng và thụt đầu dòng

Trong Python khoảng trắng rất quan trọng. Chúng ta phân biệt cách sử dụng dấu khoảng trắng. Khoảng trắng trong dòng đầu tiên được xem như dấu thụt đầu dòng. Nhưng nếu sử dụng sai thì sẽ báo lỗi.

Ví dụ như sau:

```
>>> a = 12
>>>  a = 12
File "<stdin>", line 1
a = 12
IndentationError: unexpected indent
```

```
  ^
```

Chú ý: Có một dấu cách đầu tiên gây ra lỗi trong đoạn code trên, do đó chúng ta cần đặt dấu thụt đầu dòng thích hợp.

Chúng ta có một số chú ý cho việc sử dụng dấu khoảng trắng và định danh

```
Use 4 spaces for indentation.
Never mix tab and spaces.
One blank line between functions.
Two blank lines between classes.
```

Có nhiều nơi trong đoạn code mà bạn phải áo dụng cách dùng dấu cách, ví dụ như:

```
Add a space after ”,” in dicts, lists, tuples, and argument lists and after ”:” in dicts.
Spaces around assignments and comparisons (except in argument list)
No spaces just inside parentheses.
```

Comments Comments là một đoạn văn bản được viết trong code, chúng được sử dụng để giải thích hoặc chú thích cho người khác hiểu về đoạn code đó. Một đoạn comment bắt đầu bằng dấu #, mọi thứ sau dấu comment không được thực thi trong chương trình.

```
>>> # This is a comment
>>> # The next line will add two numbers
>>> a = 12 + 34
>>> print c #this is a comment too :)
```

Comments giúp cho lập trình viên dễ dàng cải tiến mã nguồn, ghi chú các chức năng của đoạn code thực hiện, nó có thể là người viết, ngày viết.

```
# FIXME -- fix these code later
# TODO -- in future you have to do this
```

Modules Modules trong Python là các tập tin chưa các hàm được định nghĩa sẵn, biến cái mà chúng ta có thể sử dụng lại, nó cũng có đuôi mở rộng là .py. Python đã cung cấp sẵn một số module mặc định. Chúng ta có thể sử dụng chúng. Để sử dụng chúng ta cần dùng lệnh import. Ví dụ như sau:

```
>>> import math
>>> print math.e
2.71828182846
```

Chúng ta sẽ tìm hiểu chi tiết về các Module trong các phần sau. Chúc các bạn thành công!

Biến và kiểu dữ liệu

Từ khóa và định danh

Code Python có thể được chia thành các định danh. Đinh danh (hay còn được gọi là tên) được mô tả bởi các định nghĩa từ vựng sau đây:

```
identifier ::= (letter|"_") (letter | digit | "_")*
letter ::= lowercase | uppercase
lowercase ::= "a"..."z"
uppercase ::= "A"..."Z"
digit ::= "0"..."9"
```

Sau đây là danh sách các từ khóa của ngôn ngữ, chúng ta không thể sử dụng chúng như một định danh thông thường. Chúng ta phải dùng chính xác các từ sau đây:

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

Trong Python chúng ta không chỉ định một kiểu dữ liệu trong một biến. Tuy nhiên chúng ta có thể viết abc = 1 và abc sẽ trở thành dữ liệu kiểu nguyên. Nếu viết abc = 1.0 thì abc sẽ là kiểu số thực. Sau đây là ví dụ cơ bản về việc gán giá trị cho biến:

```
>>> a = 13
>>> b = 23
>>> a + b
36
```

Trong ví dụ trên bạn đã hiểu cách khai báo biến trong Python, bạn chỉ cần gõ tên và giá trị của biến. Python có thể thao tác trên chuỗi. Chúng được đặt trong dấu ngoặc đơn hoặc ngoặc kép như:

```
>>> 'India'
'India'
>>> 'India\'s best'
"India's best"
>>> "Hello World!"
'Hello World!'
```

Nhận dữ liệu từ bàn phím:

Trong Python bạn không cần bước đọc dữ liệu từ bàn phím. Bạn có thể sử dụng hàm raw_input để thực hiện nhập dữ liệu: Ví dụ như:

```
raw_input("Xin chao Python Viet Nam")
```

và sẽ trả về chuỗi đầu vào. Chúng ta sẽ thử viết một chương trình đơn giản cho phép đọc số được nhập từ bàn phím sau đó kiểm tra nó lớn hơn 100 hoặc không. Tên chương trình này có tên là testhundred.py:

```
#!/usr/bin/env python
number = int(raw_input("Enter an integer: "))
if number < 100:
    print "Your number is smaller than 100"
else:
    print "Your number is greater than 100"
```

Kết quả sẽ như sau:

```
$ ./testhundred.py
Enter an integer: 13
Your number is smaller than 100
$ ./testhundred.py
Enter an integer: 123
Your number is greater than 100
```

Trong chương trình tiếp chúng ta sẽ lấy ví dụ về việc tính toán các khoản đầu tư:

```
#!/usr/bin/env python
amount = float(raw_input("Enter amount: "))
inrate = float(raw_input("Enter Interest rate: "))
period = int(raw_input("Enter period: "))
value = 0
year = 1
while year <= period:
    value = amount + (inrate * amount)
    print "Year %d Rs. %.2f" % (year, value)
    amount = value
    year = year + 1
```

Kết quả như sau:

```
$ ./investment.py
Enter amount: 10000
Enter Interest rate: 0.14
Enter period: 5
Year 1 Rs. 11400.00
Year 2 Rs. 12996.00
Year 3 Rs. 14815.44
Year 4 Rs. 16889.60
Year 5 Rs. 19254.15
```

Một vài ví dụ:

Một vài ví dụ về biến và kiểu dữ liệu:

Trung bình của số N

Trong chương trình này chúng ta sẽ tính toán trung bình của số N

```
#!/usr/bin/env python
N = 10
sum = 0
count = 0
while count < N:
    number = float(raw_input(""))
    sum = sum + number
    count = count + 1
average = float(sum)/N
print "N = %d , Sum = %f" % (N, sum)
print "Average = %f" % average
```

Kết quả như sau

```
$ ./averagen.py
1
2.3
4.67
1.42
7
3.67
4.08
2.2
4.25
8.21
N = 10 , Sum = 38.800000
Average = 3.880000
```

Chuyển đổi nhiệt độ: Chương trình phục vụ việc chuyển đổi từ độ F sang độ C sử dụng công thức: C=(F-32)/1.8

```
#!/usr/bin/env python
fahrenheit = 0.0
print "Fahrenheit Celsius"
while fahrenheit <= 250:
    celsius = ( fahrenheit - 32.0 ) / 1.8 # Here we calculate the Celsius value
    print "%5.1f %7.2f" % (fahrenheit , celsius)
    fahrenheit = fahrenheit + 25
```

Kết quả như sau:

```
[kd@kdlappy book]$ ./temperature.py
Fahrenheit Celsius
0.0  -17.78
25.0   -3.89
50.0   10.00
75.0   23.89
100.0   37.78
125.0   51.67
150.0   65.56
175.0   79.44
200.0   93.33
225.0  107.22
250.0  121.11
```

Gán nhiều giá trị cho biến trên một dòng:

Bạn có thể khai báo nhiều giá trị cho nhiều biến trên một dòng, ví dụ như sau:

```
>>> a , b = 45, 54
>>> a
45
>>> b
54
```

Bạn cũng có thể chuyển đổi giữa hai giá trị rất dễ dàng:

```
>>> a, b = b , a
>>> a
54
>>> b
45
```

Để hiểu hơn cách làm việc, chúng ta sẽ tìm hiểu về kiểu dữ liệu tuple. Chúng ta sẽ sử dụng dấu phẩy cho việc tạo tuple. Ở bên phải chúng ta sẽ tạo một tuple (chúng ta gọi chúng là gói tuple) và phía bên trái chúng ta gọi là tuple giải nén cho một tuple mới.

Dưới đây là một ví dụ:

```
>>> data = ("Kushal Das", "India", "Python")
>>> name, country, language = data
>>> name
'Kushal Das'
>>> country
'India'
>>> language
'Python'
```

Chúc các bạn thành công!

Toán tử và biểu thức

Trong Python bạn có thể viết biểu thức trên các dòng.

Ví dụ:

```
>>> 2 + 3
5
>>> 23 - 3
20
>>> 22.0 / 12
1.8333333333333333
```

Để lấy giá trị thập phân bạn thực hiện như trên. Để lấy giá trị làm trong bạn sử dụng toán tử %

```
>>> 14 % 3
2
```

Ví dụ về số nguyên

```
#!/usr/bin/env python
days = int(raw_input("Enter days: "))
months = days / 30
days = days % 30
print "Months = %d Days = %d" % (months, days)
```

Kết quả như sau:

```
$ ./integer.py
Enter days: 265
Months = 8 Days = 25
```

Trong dòng đầu tiên chúng ta sẽ lấy số ngày, sau đó lấy số tháng và ngày và cuối cùng là in chúng ra màn hình. Bạn có thể làm chúng một cách dễ dàng.

```
#!/usr/bin/env python
days = int(raw_input("Enter days: "))
print "Months = %d Days = %d" % (divmod(days, 30))
```

Hàm divmod(num1, num2) trả về hai giá trị , first is the division of num1 and num2 and in second the modulo of num1 and num2.

Relational Operators

You can use the following operators as relational operators Relational Operators Operator Meaning

< Is less than <= Is less than or equal to

> Is greater than

> = Is greater than or equal to == Is equal to != Is not equal to

```
Ví dụ:

>>> 1 < 2
True
>>> 3 > 34
False
>>> 23 == 45
False
>>> 34 != 323
True

//

>>> 4.0 // 3
1.0
>>> 4.0 / 3
1.3333333333333333
```

Logical Operators

To do logical AND , OR we use and ,or keywords. x and y returns False if x is False else it returns evaluation of y. If x is True, it returns True.

```
>>> 1 and 4
4
>>> 1 or 4
1
>>> -1 or 4
-1
>>> 0 or 4
4
```

Shorthand Operator

x op = expression is the syntax for shorthand operators. It will be evaluated like x = x op expression , Few examples are

```
>>> a = 12
>>> a += 13
>>> a
25
>>> a /= 3
>>> a
8
>>> a += (26 * 32)
>>> a
840
```

```
shorthand.py example

#!/usr/bin/env python
N = 100
a = 2
while a < N:
    print "%d" % a
    a *= a
```

The output

```
$ ./shorthand.py
```

```
2
4
16
```

Expressions

Generally while writing expressions we put spaces before and after every operator so that the code becomes clearer to read, like

```
a = 234 * (45 - 56.0 / 34)
```

One example code used to show expressions

```
#!/usr/bin/env python
a = 9
b = 12
c = 3
x = a - b / 3 + c * 2 - 1
y = a - b / (3 + c) * (2 - 1)
z = a - (b / (3 + c) * 2) - 1
print "X = ", x
print "Y = ", y
print "Z = ", z
```

The output

```
$ ./evaluationexp.py
X =  10
Y =  7
Z =  4
```

At first x is being calculated. The steps are like this

```
9 - 12 / 3 + 3 * 2 -1
9 - 4 + 3 * 2 - 1
9 - 4 + 6 - 1
5 + 6 - 1
11 - 1
10
```

Now for y and z we have parentheses, so the expressions evaluated in different way. Do the calculation yourself to check them.

Chuyển đổi kiểu

Chúng ta có thể thực hiện chuyển đổi thủ công. Ví dụ như:

```
float(string) -> float value
int(string) -> integer value
str(integer) or str(float) -> string representation
>>> a = 8.126768
>>> str(a)
'8.126768'

evaluateequ.py
```

Sau đây là chương trình tính toán giá trị của dãy số 1/x+1/(x+1)+1/(x+2)+ ... +1/n với n tăng dần, trong trường hợp x = 1 và n =10

```
#!/usr/bin/env python
sum = 0.0
for i in range(1, 11):
    sum += 1.0 / i
print "%2d %6.4f" % (i , sum)
```

Kết quả như sau:

```
$ ./evaluateequ.py
1 1.0000
2 1.5000
3 1.8333
4 2.0833
5 2.2833
6 2.4500
7 2.5929
8 2.7179
9 2.8290
10 2.9290
```

Dòng code sum += 1.0 / i nó có nghĩa là sum = sum + 1.0 / i.

quadraticequation.py

Sau đây là chương trình giải phương trình bậc 2

```
#!/usr/bin/env python
import math
a = int(raw_input("Enter value of a: "))
b = int(raw_input("Enter value of b: "))
c = int(raw_input("Enter value of c: "))
d = b * b - 4 * a * c
if d < 0:
    print "ROOTS are imaginary"
else:
    root1 = (-b + math.sqrt(d)) / (2.0 * a)
    root2 = (-b - math.sqrt(d)) / (2.0 * a)
    print "Root 1 = ", root1
    print "Root 2 = ", root2

salesmansalary.py
```

Trong ví dụ này chúng ta sẽ thực hiện tính toán lương của một nhân viên kinh doanh camera. Lương cơ bản là 1500, với mỗi một camera bán được anh ấy sẽ có 200 và hoa hồng của nhân viên kinh doanh là 2%. Đầu vào là số camera và tổng số giá của camera.

```
#!/usr/bin/env python
basic_salary = 1500
bonus_rate = 200
commision_rate = 0.02
numberofcamera = int(raw_input("Enter the number of inputs sold: "))
price = float(raw_input("Enter the total prices: "))
bonus = (bonus_rate * numberofcamera)
commision = (commision_rate * numberofcamera * price)
print "Bonus        = %6.2f" % bonus
print "Commision    = %6.2f" % commision
print "Gross salary = %6.2f" % (basic_salary + bonus + commision)
```

Kết quả như sau:

```
$ ./salesmansalary.py
Enter the number of inputs sold: 5
Enter the total prices: 20450
Bonus        = 1000.00
Commision    = 2045.00
Gross salary = 4545.00
```

Câu lệnh điều khiển If else

Mẫu câu điều khiển như sau:

```
If statement

The syntax looks like

if expression:
    do this
```

Dưới đây là ví dụ để check một số lớn hơn hoặc nhỏ hơn 100:

```
#!/usr/bin/env python
number = int(raw_input("Enter a number: "))
if number < 100:
    print "The number is less than 100"
```

Sau đó chạy nó như sau:

```
$ ./number100.py
Enter a number: 12
The number is less than 100
```

Else

Trong trường hợp muốn in ra số đó lớn hơn 100, trường hợp này chúng ta sử dụng else. Nó sẽ thực hiện khi biểu thức if không hoàn thành.

```
#!/usr/bin/env python
number = int(raw_input("Enter a number: "))
if number < 100:
    print "The number is less than 100"
else:
    print "The number is greater than 100"
```

Kết quả như sau:

```
$ ./number100.py
Enter a number: 345
The number is greater than 100
```

Một ví dụ rất đơn giản như sau:

```
>>> x = int(raw_input("Please enter an integer: "))
>>> if x < 0:
...      x = 0
...      print 'Negative changed to zero'
... elif x == 0:
...      print 'Zero'
... elif x == 1:
...      print 'Single'
... else:
...      print 'More'
```

Kiểm tra giá trị thật

```
if x:
    pass

Warning

Don’t do this

if x == True:
    pass
```

Vòng lặp

Trong ví dụ trước, đôi khi nó được yêu cầu làm vài công việc cùng một lúc. Chúng ta sẽ phải thực hiện code nhiều lần cho việc thực thi. Nói một cách kỹ thuật thì đây là vòng lặp. Trước tiên chúng ta sẽ xem biểu thức Wile cho vòng lặp.

Vòng lặp While

Cú pháp của vòng lặp While như sau:

```
while condition:
    statement1
    statement2
```

Đoạn code chúng ta muốn chạy sau vòng lặp while phải được thụt đầu dòng. Chúng ta sẽ thực hiện chúng nếu biểu thức là đúng. Chúng ta sẽ viết đoạn code in ra số từ 0 đến 10 như sau:

```
>>> n = 0
>>> while n < 11:
...     print n
...     n += 1
...
0
1
2
3
4
5
6
7
8
9
10
```

Trong dòng đầu tiên ta khai báo n = 0, sau đó trong biểu thức while khi n < 11, điều đó có nghĩa là chúng sẽ được thực hiện đến khi n < 11. Trong vòng lặp chúng ta in giá trị của n sau đó sắp sếp chúng.

Dãy số Fibonacci Chúng ta sẽ giải bài toán Fibonacci. Chương trình như sau:

```
#!/usr/bin/env python
a, b = 0, 1
while b < 100:
    print b
    a, b = b, a + b
```

Kết quả như sau:

```
$ ./fibonacci1.py
1
1
2
3
5
8
13
21
34
55
89
```

Trong dòng đầu tiên của đoạn code chúng ta sẽ gán a và b, sau đó vòng lặp while với b có giá trị nhỏ hơn 100. Trong lặp đầu tiên chúng ta sẽ in ra giá trị b sau đó tiếp theo sẽ in giá trị của b cho a và a +b cho b trong dòng tiếp theo.

```
#!/usr/bin/env python
a, b = 0, 1
while b < 100:
    print b,
    a, b = b, a + b
```

Kết quả sẽ như sau:

```
$ ./fibonacci2.py
1 1 2 3 5 8 13 21 34 55 89
```

Power Series

Chúng ta sẽ viết chương trình để tính toán ví dụ: e**x =1+x+x**2/2! +x**3/3! +....+ x**n/n! where 0 < x < 1

```
#!/usr/bin/env python
x = float(raw_input("Enter the value of x: "))
n = term = num = 1
sum = 1.0
while n <= 100:
    term *= x / n
    sum += term
    n += 1
    if term < 0.0001:
        break
print "No of Times= %d and Sum= %f" % (n, sum)
```

Kết quả như sau:

```
$ ./powerseries.py
Enter the value of x: 0
No of Times= 2 and Sum= 1.000000
$ ./powerseries.py
Enter the value of x: 0.1
No of Times= 5 and Sum= 1.105171
$ ./powerseries.py
Enter the value of x: 0.5
No of Times= 7 and Sum= 1.648720
```

Trong chương trình chúng ta có một từ khóa mới được gọi là break. Từ khóa sẽ thực hiện dừng khi vòng lặp đang chạy. Ví dụ như sau:

```
if term < 0.0001:
    break
```

Điều nayfcos nghĩa nếu giá trị của term nhỏ hơn 0.0001 sau đó thoát khỏi vòng lặp.

Phép nhân bảng:

Trong ví dụ sau chúng ta sẽ in ra bảng phéo nhân tới 10

```
#!/usr/bin/env python
i = 1
print "-" * 50
while i < 11:
    n = 1
    while n <= 10:
        print "%4d" % (i * n),
        n += 1
    print ""
    i += 1
print "-" * 50
```

Kết quả như sau:

```
$ ./multiplication.py
--------------------------------------------------
   1    2    3    4    5    6    7    8    9   10
   2    4    6    8   10   12   14   16   18   20
   3    6    9   12   15   18   21   24   27   30
   4    8   12   16   20   24   28   32   36   40
   5   10   15   20   25   30   35   40   45   50
   6   12   18   24   30   36   42   48   54   60
   7   14   21   28   35   42   49   56   63   70
   8   16   24   32   40   48   56   64   72   80
   9   18   27   36   45   54   63   72   81   90
  10   20   30   40   50   60   70   80   90  100
--------------------------------------------------
```

Trong hàm print ở trên chúng ta in chuỗi với số lần là n, chuỗi sẽ được xuất hiện n lần. Ví dụ:

```
>>> print "*" * 10
**********
>>> print "#" * 20
####################
>>> print "--" * 20
----------------------------------------
>>> print "-" * 40
----------------------------------------
```

Some printing * examples

Here are some examples which you can find very often in college lab reports

Design 1

```
#!/usr/bin/env python
row = int(raw_input("Enter the number of rows: "))
n = row
while n >= 0:
    x =  "*" * n
    print x
    n -= 1
```

Kết quả như sau

```
$ ./design1.py
Enter the number of rows: 5
*****
****
***
**
```

```
*
```

Design 2

```
#!/usr/bin/env python
n = int(raw_input("Enter the number of rows: "))
i = 1
while i <= n:
    print "*" * i
    i += 1

The output
```

```
$ ./design2.py
Enter the number of rows: 5
*
**
***
****
*****
```

Design 3

```
#!/usr/bin/env python
row = int(raw_input("Enter the number of rows: "))
n = row
while n >= 0:
    x = "*" * n
    y = " " * (row - n)
    print y + x
    n -= 1
```

Kết quả:

```
$ ./design3.py
Enter the number of rows: 5
*****
 ****
  ***
   **
    *
```

Danh sách

Danh sách là kiểu dữ liệu có cấu trúc.

Chúng ta sẽ tìm hiểu về một kiểu dữ liệu có cấu trúc trước khi chúng ta tìm hiểu thêm về vòng lặp. Danh sách có thể viết như một danh sách các giá trị cách nhau bởi dấu phẩy (,) và nằm trong dấu ngoặc vuông.

```
>>> a = [ 1, 342, 2233423, 'India', 'Fedora']
>>> a
[1, 342, 2233423, 'India', 'Fedora']
```

Danh sách có thể chứa bất cứ dữ liệu nào trong chúng. Nó hoạt động liên tiếp có nghĩa là

```
>>> a[0]
```

```
1
>>> a[4]
'Fedora'
```

Bạn có thể chia chúng thành nhiều phần khác nhau ví dụ:

```
>>> a[4]
'Fedora'
>>> a[-1]
'Fedora'
>>> a[-2]
'India'
>>> a[0:-1]
[1, 342, 2233423, 'India']
>>> a[2:-2]
[2233423]
>>> a[:-2]
[1, 342, 2233423]
>>> a[0::2]
[1, 2233423, 'Fedora']
```

Trong ví dụ này dấu : đại diện cho khoảng dữ liệu, cụ thể s[i:j:k] có nghĩa là giá trị s từ j tới j sau đó đến k. Để kiểm tra nếu giá trị có hoặc tồn tại trong danh sách hoặc không bạn có thể dùng

```
>>> a = ['Fedora', 'is', 'cool']
>>> 'cool' in a
True
>>> 'Linux' in a
False
```

Điều đó có nghĩa là bạn có thể sử dụng biểu thức như một mệnh đề if. Hàm len() cung cấp cho chúng ta độ dài của danh sách.

```
>>> len(a)
3
```

Ghi chú

Nếu bạn kiểm tra nếu danh sách rỗng hoặc không, bạn có thể làm như sau:

```
if list_name: #This means the list is not empty
    pass
else: #This means the list is empty
    pass

For loop
```

Đây cũng là một dạng vòng lặp sử dụng biểu thức. Trong Python biểu thức có cách làm việc khác với trong C. Ví dụ

```
>>> a = ['Fedora', 'is', 'powerfull']
>>> for x in a:
...     print x,
...
Fedora is powerfull
```

Chúng ta cũng có thể dùng như sau:

```
>>> a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> for x in a[::2]:
...     print x
...
1
3
5
7
9
```

Hàm range()

Là một hàm được cung cấp sẵn trong Python ví dụ

range(...) range([start,] stop[, step]) -> danh sách số nguyên

Trả về danh sách các số nguyên. range(i, j) trả về [i, i+1, i+2, ..., j-1]; bắt đầu (!) được gán mặc định là 0.

Ở bước đưa ra, nó chỉ định tăng dần hoặc giảm dần. Ví dụ, range(4) sẽ trả về giá trị [0, 1, 2, 3]. Điểm cuối cùng được bỏ qua. Đây là các giá trị chính xác của danh sách gồm 4 phần tử.

Giờ nếu chúng ta muốn nhìn thấy thông điệp giữ đỡ của hệ thống bạn có thể sử dụng lệnh help(range) trong trình thông dịch Python, lệnh help(s) sẽ trả về dòng hướng dẫn trong một object s. Ví dụ với hàm range

```
>>> range(1, 5)
[1, 2, 3, 4]
>>> range(1, 15, 3)
[1, 4, 7, 10, 13]
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Continue statement Nếu như chúng ta có lệnh dừng biểu thức break, thì cũng sẽ có continue, cái mà sẽ bỏ qua việc thực thi đoạn code và quay trở lại điểm xuất phát của vòng lặp. Điều đó có nghĩa nó sẽ giúp bạn bỏ qua một phần vòng lặp. Trong ví dụ dưới đây chúng ta sẽ thực hiện lại.

```
#!/usr/bin/env python
while True:
    n = int(raw_input("Please enter an Integer: "))
    if n < 0:
        continue #this will take the execution back to the starting of the loop
    elif n == 0:
        break
    print "Square is ", n ** 2
print "Goodbye"
```

Kết quả là

```
$ ./continue.py
Please enter an Integer: 34
Square is 1156
Please enter an Integer: 4
Square is 16
Please enter an Integer: -9
Please enter an Integer: 0
Goodbye
```

Vòng lặp Else

Chúng ta có tùy chọn else của biểu thức sau bất cứ vòng lặp nào. Nó sẽ được thực hiện sau khi vòng lặp dừng lại.

```
>>> for i in range(0, 5):
...     print i
... else:
...     print "Bye bye"
...
0
1
2
3
4
Bye bye
```

Game of sticks

```
#!/usr/bin/env python
sticks = 21

print "There are 21 sticks, you can take 1-4 number of sticks at a time."
print "Whoever will take the last stick will loose"

while True:
    print "Sticks left: " , sticks
    sticks_taken = int(raw_input("Take sticks(1-4):"))
    if sticks == 1:
        print "You took the last stick, you loose"
        break
    if sticks_taken >= 5 or sticks_taken <= 0:
        print "Wrong choice"
        continue
    print "Computer took: " , (5 - sticks_taken) , "n\n"
    sticks -= 5
```

Cấu trúc dữ liệu

Python có một vài cấu trúc dữ liệu có sẵn. Nếu bạn chỉ tự hỏi cấu trúc dữ liệu là gì, sau đó không có một cách nào để lưu trữ dữ liệu và các phương pháp cụ thể để lấy lại hoặc chỉnh sửa nó. Chúng ta đã nghiên cứu về List và trong phần này chúng ta sẽ đi nghiên cứu sâu hơn

Lists

```
>>> a = [23, 45, 1, -3434, 43624356, 234]
>>> a.append(45)
>>> a
[23, 45, 1, -3434, 43624356, 234, 45]
```

Đầu tiên chúng ta sẽ tạo một danh sách a. Sau đó sẽ thêm giá trị 45 vào cuối của danh sách chúng ta sử dụng phương thức a.append(45). Bạn sẽ thấy 45 được gán vào cuối của dãy. Đôi khi nó cũng có thể đòi hỏi chèn dữ liệu ở bất cứ nơi nào của List (danh sách), bạn sử dụng hàm insert()

```
>>> a.insert(0, 1) # 1 added at the 0th position of the list
>>> a
[1, 23, 45, 1, -3434, 43624356, 234, 45]
>>> a.insert(0, 111)
>>> a
[111, 1, 23, 45, 1, -3434, 43624356, 234, 45]
```

count(s) will return you number of times s is in the list. Here we are going to check how many times 45 is there in the list. count(s) sẽ trả về số lần s trong list. Chúng ta sẽ đi kiểm tra xem số lần số 45 xuất hiện trong danh sách.

```
>>> a.count(45)
2
```

If you want to remove any particular value from the list you have to use remove() method. Nếu bạn muốn bỏ bất cứ giá trị nào từ trong list bạn có thể sử dụng hàm remove().

```
>>> a.remove(234)
>>> a
[111, 1, 23, 45, 1, -3434, 43624356, 45]
```

Chúng ta cũng có thể lấy lại bằng phương thức, ví du:

```
>>> a.reverse()
>>> a
[45, 43624356, -3434, 1, 45, 23, 1, 111]
```

Chúng ta có thể lưu trữ bất cứ gì trong danh sách, đầu tiên chúng ta sẽ thêm một danh sách b vào trong a, sau đó chúng ta sẽ nghiên cứu cách để thêm giá trị của b vào trong a.

```
>>> b = [45, 56, 90]
>>> a.append(b)
>>> a
[45, 43624356, -3434, 1, 45, 23, 1, 111, [45, 56, 90]]
>>> a[-1]
```

```
[45, 56, 90]
>>> a.extend(b) #To add the values of b not the b itself
>>> a
[45, 43624356, -3434, 1, 45, 23, 1, 111, [45, 56, 90], 45, 56, 90]
>>> a[-1]
90
```

Ở trên bạn có thể thấy chúng ta sử dụng a.extend() để mở rộng danh sách. Việc sắp xếp lại bạn dùng hàm sort()

```
>>> a.sort()
>>> a
[-3434, 1, 1, 23, 45, 45, 45, 56, 90, 111, 43624356, [45, 56, 90]]
```

Bạn cũng có thể xóa giá trị của bất cứ vị trí nào của danh sách bằng cách sử dụng lệnh del.

```
>>> del a[-1]
>>> a
[-3434, 1, 1, 23, 45, 45, 45, 56, 90, 111, 43624356]
```

Sử dụng danh sách như ngăn xếp và hàng đợi Ngăn xếp thường được biết đến như LIFO (Last In First Out). Nó có nghĩa là dữ liệu sẽ được đưa vào phần cuối, sau đó dữ liệu cuối cùng sẽ đi ra đầu.

```
>>> a
[1, 2, 3, 4, 5, 6]
>>> a.pop()
6
>>> a.pop()
5
>>> a.pop()
4
>>> a.pop()
3
>>> a
[1, 2]
>>> a.append(34)
>>> a
[1, 2, 34]
```

We learned a new method above pop(). pop(i) will take out the ith data from the list.

In our daily life we have to encounter queues many times, like in ticket counters or in library or in the billing section of any supermarket. Queue is the data structure where you can append more data at the end and take out data from the beginning. That is why it is known as FIFO (First In First Out).

```
>>> a = [1, 2, 3, 4, 5]
>>> a.append(1)
>>> a
[1, 2, 3, 4, 5, 1]
>>> a.pop(0)
1
>>> a.pop(0)
2
>>> a
[3, 4, 5, 1]
```

To take out the first element of the list we are using a.pop(0). List Comprehensions

List comprehensions provide a concise way to create lists. Each list comprehension consists of an expression followed by a for clause, then zero or more for or if clauses. The result will be a list resulting from evaluating the expression in the context

of the for and if clauses which follow it.

For example if we want to make a list out of the square values of another list, then

```
>>> a = [1, 2, 3]
>>> [x ** 2 for x in a]
[1, 4, 9]
>>> z = [x + 1 for x in [x ** 2 for x in a]]
>>> z
[2, 5, 10]
```

Above in the second case we used two list comprehensions in a same line. Tuples

Tuples are data separated by comma.

```
>>> a = 'Fedora', 'Debian', 'Kubuntu', 'Pardus'
>>> a
('Fedora', 'Debian', 'Kubuntu', 'Pardus')
>>> a[1]
'Debian'
>>> for x in a:
...     print x,
...
```

Fedora Debian Kubuntu Pardus

You can also unpack values of any tuple in to variables, like

```
>>> divmod(15,2)
(7, 1)
>>> x, y = divmod(15,2)
>>> x
7
>>> y
1
```

Tuples are immutable, that means you can not del/add/edit any value inside the tuple. Here is another example

```
>>> a = (1, 2, 3, 4)
>>> del a[0]
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: 'tuple' object doesn't support item deletion
```

Above you can see Python is giving error when we are trying to delete a value in the tuple.

To create a tuple which contains only one value you have to type a trailing comma.

```
>>> a = (123)
>>> a
123
>>> type(a)
<type 'int'>
>>> a = (123, ) #Look at the trailing comma
>>> a
(123,)
>>> type(a)
<type 'tuple'>
```

Using the built in function type() you can know the data type of any variable. Remember the len() function we used to find the length of any sequence ?

```
>>> type(len)
<type 'builtin_function_or_method'>
```

Sets

Sets are another type of data structure with no duplicate items. We can also mathematical set operations on sets.

```
>>> a = set('abcthabcjwethddda')
>>> a
set(['a', 'c', 'b', 'e', 'd', 'h', 'j', 't', 'w'])
```

And some examples of the set operations

```
>>> a = set('abracadabra')
>>> b = set('alacazam')
>>> a                                  # unique letters in a
set(['a', 'r', 'b', 'c', 'd'])
>>> a - b                              # letters in a but not in b
set(['r', 'd', 'b'])
>>> a | b                              # letters in either a or b
set(['a', 'c', 'r', 'd', 'b', 'm', 'z', 'l'])
>>> a & b                              # letters in both a and b
set(['a', 'c'])
>>> a ^ b                              # letters in a or b but not both
set(['r', 'd', 'b', 'm', 'z', 'l'])
```

To add or pop values from a set

```
>>> a
set(['a', 'c', 'b', 'e', 'd', 'h', 'j', 'q', 't', 'w'])
>>> a.add('p')
>>> a
set(['a', 'c', 'b', 'e', 'd', 'h', 'j', 'q', 'p', 't', 'w'])
```

Dictionaries

Dictionaries are unordered set of key: value pairs where keys are unique. We declare dictionaries using {} braces. We use dictionaries to store data for any particular key and then retrieve them.

```
>>> data = {'kushal':'Fedora', 'kart_':'Debian', 'Jace':'Mac'}
>>> data
{'kushal': 'Fedora', 'Jace': 'Mac', 'kart_': 'Debian'}
>>> data['kart_']
'Debian'
```

We can add more data to it by simply

```
>>> data['parthan'] = 'Ubuntu'
>>> data
{'kushal': 'Fedora', 'Jace': 'Mac', 'kart_': 'Debian', 'parthan': 'Ubuntu'}
```

To delete any particular key:value pair

```
>>> del data['kushal']
>>> data
{'Jace': 'Mac', 'kart_': 'Debian', 'parthan': 'Ubuntu'
```

To check if any key is there in the dictionary or not you can use in keyword.

```
>>> 'Soumya' in data
False
```

You must remember that no mutable object can be a key, that means you can not use a list as a key.

dict() can create dictionaries from tuples of key,value pair.

```
>>> dict((('Indian','Delhi'),('Bangladesh','Dhaka')))
{'Indian': 'Delhi', 'Bangladesh': 'Dhaka'}
```

If you want to loop through a dict use iteritems() method.

```
>>> data
{'Kushal': 'Fedora', 'Jace': 'Mac', 'kart_': 'Debian', 'parthan': 'Ubuntu'}
>>> for x, y in data.iteritems():
...     print "%s uses %s" % (x, y)
...
Kushal uses Fedora
Jace uses Mac
kart_ uses Debian
parthan uses Ubuntu
```

Many times it happens that we want to add more data to a value in a dictionary and if the key does not exists then we add some default value. You can do this efficiently using dict.setdefault(key, default).

```
>>> data = {}
>>> data.setdefault('names', []).append('Ruby')
>>> data
{'names': ['Ruby']}
>>> data.setdefault('names', []).append('Python')
>>> data
{'names': ['Ruby', 'Python']}
>>> data.setdefault('names', []).append('C')
>>> data
{'names': ['Ruby', 'Python', 'C']}
```

When we try to get value for a key which does not exists we get KeyError. We can use dict.get(key, default) to get a default value when they key does not exists before.

```
>>> data['foo']
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
KeyError: 'foo'
>>> data.get('foo', 0)
0
```

If you want to loop through a list (or any sequence) and get iteration number at the same time you have to use enumerate().

```
>>> for i, j in enumerate(['a', 'b', 'c']):
...     print i, j
...
0 a
1 b
2 c
```

You may also need to iterate through two sequences same time, for that use zip() function.

```
>>> a = ['Pradeepto', 'Kushal']
>>> b = ['OpenSUSE', 'Fedora']
>>> for x, y in zip(a, b):
...     print "%s uses %s" % (x, y)
...
Pradeepto uses OpenSUSE
Kushal uses Fedora

students.py
```

In this example , you have to take number of students as input , then ask marks for three subjects as ‘Physics’, ‘Maths’, ‘History’, if the total marks for any student is less 120 then print he failed, or else say passed.

```
#!/usr/bin/env python
n = int(raw_input("Enter the number of students:"))
data = {} # here we will store the data
languages = ('Physics', 'Maths', 'History') #all languages
for i in range(0, n): #for the n number of students
    name = raw_input('Enter the name of the student %d: ' % (i + 1)) #Get the name of the student
    marks = []
    for x in languages:
        marks.append(int(raw_input('Enter marks of %s: ' % x))) #Get the marks for  languages
    data[name] = marks
for x, y in data.iteritems():
    total =  sum(y)
    print "%s 's  total marks %d" % (x, total)
    if total < 120:
        print "%s failed :(" % x
    else:
        print "%s passed :)" % x
```

The output

```
[kd@kdlappy book]$ ./students.py
Enter the number of students:2
Enter the name of the student 1: Babai
Enter marks of Physics: 12
Enter marks of Maths: 45
Enter marks of History: 40
Enter the name of the student 2: Tesla
Enter marks of Physics: 99
Enter marks of Maths: 98
Enter marks of History: 99
Babai 's  total marks 97
Babai failed :(
Tesla 's  total marks 296
Tesla passed :)

matrixmul.py
```

In this example we will multiply two matrices. First we will take input the number of rows/columns in the matrix (here we assume we are using n x n matrix). Then values of the matrices.

```
#!/usr/bin/env python
n = int(raw_input("Enter the value of n: "))
print "Enter values for the Matrix A"
a = []
for i in range(0, n):
    a.append([int(x) for x in raw_input("").split(" ")])
print "Enter values for the Matrix B"
b = []
for i in range(0, n):
    b.append([int(x) for x in raw_input("").split(" ")])
c = []
for i in range(0, n):
    c.append([a[i][j] * b[j][i] for j in range(0,n)])
print "After matrix multiplication"
print "-" * 10 * n
for x in c:
    for y in x:
        print "%5d" % y,
    print ""
print "-" * 10 * n
```

The output

```
[kd@kdlappy book]$ ./matrixmul.py
Enter the value of n: 3
Enter values for the Matrix A
1 2 3
4 5 6
7 8 9
Enter values for the Matrix B
9 8 7
6 5 4
3 2 1
After matrix multiplication
------------------------------
    9    12     9
   32    25    12
   49    32     9
------------------------------
```

Here we have used list comprehensions couple of times. [int(x) for x in raw_input(“”).split(” ”)] here first it takes the input as string by raw_input(), then split the result by ” ”, then for each value create one int. We are also using [a[i][j] * b[j][i] for j in range(0,n)] to get the resultant row in a single line.

Chuỗi

Chuỗi là văn bản đơn gian. Trong Python chúng ta khai báo chuỗi giữa các dấu "" hoặc '' hoặc ''' hoặc """ """. Ví dụ dưới đây sẽ giúp bạn hiểu rõ hơn.

```
>>> s = "I am Indian"
>>> s
'I am Indian'
>>> s = 'I am Indian'
>>> s = "Here is a line \
... splitted in two lines"
>>> s
'Here is a line split in two lines'
>>> s = "Here is a line \n split in two lines"
>>> s
'Here is a line \n split in two lines'
>>> print s
Here is a line
split in two lines
```

Nếu bạn muốn in ra các chuỗi bạn có thể sử dụng dấu ba ngoặc/ hai ngoặc như sau:

```
>>> s = """ This is a
... multiline string, so you can
... write many lines"""
>>> print s
This is a
multiline string, so you can
write many lines
```

Các phương pháp của chuỗi.

Mỗi chuỗi có cung cấp các phương thức làm việc có sẵn, chúng ta sẽ nghiên cứu hàm s.split("")

```
>>> s = "kushal das"
>>> s.title()
'Kushal Das'
```

Trong ví dụ này hàm title() sẽ chuyển các ký tự đầu của chuỗi từ chữ thường thành chữ hoa.

Chúng ta có thể sử dụng để chuyển chuỗi thành chữ hoa hoặc chữ thường, ví dụ như sau:

```
>>> z = s.upper()
>>> z
'KUSHAL DAS'
>>> z.lower()
'kushal das'
```

upper() trả về ký tự hoa hoặc lower() trả về ký tự thường như ví dụ trên.

```
>>> s = "I am A pRoGraMMer"
>> s.swapcase()
'i AM a PrOgRAmmER'
```

swapcase() trả về chuỗi với chuỗi đã được hoán đổi so với chuỗi ban đầu.

```
>>> s = "jdwb 2323bjb"
>>> s.isalnum()
False
```

Với làm isalnum() sẽ trả về kết quả False do hàm hay sẽ kiểm tra các ký tự có phải là ký tự hay không.

```
>>> s = "jdwb2323bjb"
>>> s.isalnum()
True
```

Đây là kết quả khi không có ký tự trắng.

```
>>> s = "SankarshanSir"
>>> s.isalpha()
True
>>> s = "Sankarshan Sir"
>>> s.isalpha()
False
```

isalpha() kiểm tra ký tự trong bảng chữ cái.

```
>>> s = "1234"
>>> s.isdigit() #To check if all the characters are digits or not
True
>>> s = "Fedora9 is coming"
>>> s.islower() # To check if all chracters are lower case or not
False
>>> s = "Fedora9 Is Coming"
>>> s.istitle() # To check if it is a title or not
True
>>> s = "INDIA"
>>> s.isupper() # To check if characters are in upper case or not
True
```

Để tách chuỗi chúng ta có thể dùng làm split(). Nó sẽ trả về chuỗi được tách ra.

```
>>> s = "We all love Python"
>>> s.split(" ")
['We', 'all', 'love', 'Python']
>>> x = "Nishant:is:waiting"
>>> x.split(':')
['Nishant', 'is', 'waiting']
```

Chúng ta cũng có thể dùng làm join() để thực hiện ghép chuỗi

```
>>> "-".join("GNU/Linux is great".split(" "))
'GNU/Linux-is-great'
```

Trong ví dụ trên chúng ta chia chuỗi "GNU/Linux is great" được chia ra bằng dầu cách, và chúng ta sẽ thay thế bằng dấu "-".

Dải chuỗi

Ví dụ:

> s = '' abc\n '' s.strip() 'abc'

Bạn có thể thực hành strip từ bên phải hoặc bên trái chuỗi sử dụng hai hàm lstrip(chars) hoặc rstrip(chars).

```
>>> s = "www.foss.in"
>>> s.lstrip("cwsd.")
'foss.in'
>>> s.rstrip("cnwdi.")
'www.foss'
```

Tìm kiếm văn bản:

Python cũng cung cấp cho chúng ta hàm để tìm kiếm ký tự/ hoặc chuỗi con trong một chuỗi. Ví dụ:

```
>>> s = "faulty for a reason"
>>> s.find("for")
7
>>> s.find("fora")
-1
>>> s.startswith("fa") #To check if the string startswith fa or not
True
>>> s.endswith("reason") #
True
```

find() sẽ trả về chuỗi nếu như có kết quả và -1 nếu như không có.

Palindrome

```
#!/usr/bin/env python
s = raw_input("Please enter a string: ")
z = s[::-1]
if s == z:
    print "The string is a palindrome"
else:
    print "The string is not a palindrome"
```

Kết quả

```
$ ./palindrome.py
Please enter a string: madam1
The string is not a palindrome
$ ./palindrome.py
Please enter a string: madam
The string is a palindrome
```

Số từ

Trong ví dụ này chúng ta sẽ đếm số từ trong dòng. Ví dụ:

```
#!/usr/bin/env python
s = raw_input("Enter a line: ")
print "The number of words in the line are %d" % (len(s.split(" ")))
```

Kết quả như sau

```
$ ./countwords.py
Enter a line: Sayamindu is a great programmer
The number of words in the line are 5
```

Chúc bạn thành công!

Hàm

Trong lập trình chúng ta sẽ có lúc sử dụng một đoạn code lặp đi lặp lại nhiều lần trong chương trình. Hàm sẽ giúp chúng ta thực hiện điều này. Chúng ta có thể viết tất cả những gì chúng ta muốn thực hiện trong hàm sau đó chỉ cần gọi khi sử dụng. Như ở trên chúng ta đã sử dụng các hàm có sẵn như len(), divmod().

Định nghĩa hàm

Chúng ta sẽ dùng tư khóa *def* để thực hiện định nghĩa hàm. Ví dụ như sau:

```
def functionname(params):
    statement1
    statement2
```

Chúng ta sẽ thử viết một chương trình nhỏ để thực hiện tính tổng của hai số nguyên.

```
>>> def sum(a, b):
...     return a + b
```

ở dòng thứ hai với giá trị từ bàn phím, chúng ta sẽ trả lại kết quả của a + b. Bạn phải gọi lại nó như sau:

```
>>> res = sum(234234, 34453546464)
>>> res
34453780698L
```

Trong chương trình về số palindrom ở chương trước, chúng ta hay viết một hàm để kiểm tra chuỗi có phải là palindrome hay không sau đó trả về kết quả True hoặc False

```
#!/usr/bin/env python
def palindrome(s):
    return s == s[::-1]
if __name__ == '__main__':
    s = raw_input("Enter a string: ")
    if palindrome(s):
        print "Yes, this is  a palindrome"
    else:
        print "Oh no, not a palindrome"
```

Giờ bạn kiểm tra nào.

Biến cục bộ và biến toàn cục

Để có thể hiểu thế nào là biến cục bộ và biến toàn cục chúng ta sẽ xem hai ví dụ sau"

```
#!/usr/bin/env python
def change(b):
    a = 90
    print a
a = 9
print "Before the function call ", a
print "inside change function", change(a)
print "After the function call ", a
```

Kết quả như sau:

```
$ ./local.py
Before the function call  9
inside change function 90
After the function call  9
```

Giải thích:

Chúng ta có thể thấy, đầu tiên chúng ta sẽ gán 9 cho biến a, sau đó sẽ gọi hàm change(), trong hàm change() chúng ta sẽ gán giá trị 90 cho a và in ra số a. Sau khi hàm được gọi chúng ta sẽ in ra một lần nữa giá trị của a. Khi chúng ta gán a = 90 trong hàm, nó sẽ tạo một biến mới là biến a, nhưng biến a chỉ có giá trị và thực thi khi hàm được gọi và kết thúc khi hàm thực hiện xong. Ở đây cùng một tên nhưng chúng khác nhau khi ở trong và ngoài hàm.

```
#!/usr/bin/env python
def change(b):
    global a
    a = 90
    print a
a = 9
print "Before the function call ", a
print "inside change function",
change(a)
print "After the function call ", a
```

Ở ví dụ này chúng ta sẽ định nghĩa biến toàn cục, tuy nhiên khi thay đổi giá trị a trong hàm nó chuyển đổi ra bên ngoài của hàm.

Kết quả ví dụ trên như sau:

```
>>>
Before the function call  9
inside change function 90
After the function call  90
>>>
```

Giá trị đối số mặc định

Trong hàm biến có thể có đối số mặc định, điều đó có nghĩa nếu chúng ta không đưa bất kì giá trị nào cho biến thì nó sẽ được gán mặc định.

```
>>> def test(a , b=-99):
...     if a > b:
...         return True
...     else:
...         return False
```

Ở ví dụ trên chúng ta gán b = -99 trong hàm. Điều đó có nghĩa là giá trị của b sẽ là -99. Và nếu khi truyền giá trị cho biến của hàm nếu không truyền giá trị b thì mặc định sẽ là -99

```
>>> test(12, 23)
False
>>> test(12)
True

Important
```

Bạn hãy ghi nhớ chúng ta không thể có một đối số với đối sô mặc định nếu bạn không có một giá trị mặc định trước đó. Ví dụ f(a, b=90, c) thì a, b sẽ có giá trị còn c không có giá trị nào.

Và bạn cũng nên chú ý giá trị mặc định chỉ được tính toán một lần, nếu bạn có bất cứ sự thay đổi nào sẽ làm danh sách thay đổi. Hãy xem ví dụ sau:

```
>>> def f(a, data=[]):
...     data.append(a)
...     return data
...
>>> print f(1)
[1]
>>> print f(2)
[1, 2]
>>> print f(3)
[1, 2, 3]
```

Để tránh điều này Python cung cấp cách cho bạn như sau:

```
>>> def f(a, data=None):
...     if data is None:
...         data = []
...         data.append(a)
...         return data
...
>>> print f(1)
[1]
>>> print f(2)
[2]
```

Keyword arguments

```
>>> def func(a, b=5, c=10):
...     print 'a is', a, 'and b is', b, 'and c is', c
...
>>> func(12, 24)
a is 12 and b is 24 and c is 10
>>> func(12, c = 24)
a is 12 and b is 5 and c is 24
>>> func(b=12, c = 24, a = -1)
a is -1 and b is 12 and c is 24
```

Trong ví dụ trên bạn có thể thấy chúng ta sẽ gọi hàm với tên biến, ví dụ như func(12, c = 24), bằng cách đó chúng ta sẽ gán 24 cho c và b sẽ lấy giá trị mặc định. Bạn cũng nên chú ý trường hợp như sau:

```
>>> def func(a, b=13, v):
...     print a, b, v
...
File "<stdin>", line 1
SyntaxError: non-default argument follows default argument
```

Docstrings

Trong Python chúng ta sử dụng docstrings để giải thích sử dụng code như thế nào, nó sẽ hữu ích trong trình thông dịch và tạo tài liệu tự động. Dưới đây là ví dụ về docstring

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

```
def longest_side(a, b):
    """
    Function to find the length of the longest side of a right triangle.

    :arg a: Side a of the triangle
    :arg b: Side b of the triangle

    :return: Length of the longest side c as float
    """
    return math.sqrt(a*a + b*b)

if __name__ == '__main__':
    print longest_side(4, 5)
```

Chúng ta sẽ tìm hiểu thêm về docstrings trong chương sau.

Hàm bậc cao Hàm bậc cao hay là functor là hàm có các chức năng sau:

```
    Takes one or more functions as argument.
    Returns another function as output.
```

Ví dụ:

```
>>> def high(func, value):
...     return func(value)
...
>>> lst = high(dir, int)
>>> print lst[-3:]
['imag', 'numerator', 'real']
>>> print lst
```

map function

map rất hữu dụng trong hàm bậc cao trong Python.

Ví dụ:

```
>>> lst = [1, 2, 3, 4, 5]
>>> def square(num):
...     "Returns the square of a given number."
...     return num * num
...
>>> print map(square, lst)
[1, 4, 9, 16, 25]
```

Chúc các bạn thành công!

Xử lý tập tin

Một tập tin chứa thông tin hoặc dữ liệu được lưu trữ trên thiết bị lưu trữ của máy tính. Như bạn đã biết về các kiểu của tập tin như âm nhạc, video, và tập tin văn bản. Python sẽ cung cấp cho bạn cách để điều khiển các tập tin. Chúng ta sẽ tập trung vào hai loại: tập tin văn bản và binary.

File opening

Để thực hiện mở một tập tin chúng ta sử dụng hàm open(). Với hàm này sẽ có hai tham số được truyền vào đó là đường dẫn và chế độ mở. Các chế độ như sau:

```
    “r” -> Chế độ chỉ đọc, bạn không thể xóa hay chỉnh sửa gì.
    “w” -> Chế độ ghi, bạn có thể ghi dữ liệu vào tập tin.
    “a” -> open in append mode
```

Mặc định chế độ read sẽ đươc thiết lập, ví dụ

```
>>> fobj = open("love.txt")
>>> fobj
<open file 'love.txt', mode 'r' at 0xb7f2d968>
```

Đóng tập tin

Sau khi bạn mở tập tin để thao tác bạn nên đóng tập tin đó lại sau khi thao tác, bạn có thể sử dụng hàm close().

```
>>> fobj = open("love.txt")
>>> fobj
<open file 'love.txt', mode 'r' at 0xb7f2d968>
>>> fobj.close()
```

Ghi chú

Bạn cần chắc chắn rằng bạn luôn đóng sau khi mở một tập tin, sau khi công việc được thực hiện bạn không còn lý do nào để mở nó nữa. Bởi vì có giới hạn số tập tin mà chương trình có thể mở. Nếu bạn mở quá giới hạn sẽ không có cách nào khôi phục được, hoặc cũng có thể gặp sự cố. Với mỗi tập tin được mở tài nguyên bộ nhớ sẽ được cung cấp cho nó như mô tả tập tin xử lý hoặc khóa tập tin. Vì vậy bạn có thể tiết kiệm được tài nguyền không sử dụng hoặc ít sử dụng. Mở tập tin liên tục có thể gây ra lỗi hoặc mất dữ liệu.

Reading a file

Để đọc tập tin chúng ta có thể sử dụng hàm read().

```
>>> fobj = open("sample.txt")
>>> fobj.read()
'I love Python\nPradeepto loves KDE\nSankarshan loves Openoffice\n'
```

Nếu bạn gọi hàm read() nó sẽ trả về chuỗi trong tập tin mà nó đọc được. readline() có thể giúp bạn đọc từng dòng của tập tin. Ví dụ:

```
>>> fobj = open("sample.txt")
```

```
>>> fobj.readline()
'I love Python\n'
>>> fobj.readline()
'Pradeepto loves KDE\n'
```

Để đọc toàn bộ các dòng chúng ta dùng readlines().

```
>>> fobj = open("sample.txt")
>>> fobj.readlines()
['I love Python\n', 'Pradeepto loves KDE\n', 'Sankarshan loves Openoffice\n']
```

You can even loop through the lines in a file object.

```
>>> fobj = open("sample.txt")
>>> for x in f:
...     print x,
...
I love Python
Pradeepto loves KDE
Sankarshan loves Openoffice
```

Chúng ta sẽ viết chương trình nhỏ đưa tên của tập tin sau đó đưa ra màn hình nội dung của tập tin đó.

```
#!/usr/bin/env python
name = raw_input("Enter the file name: ")
fobj = open(name)
print fobj.read()
fobj.close()
```

Cuối chương trình chúng ta sử dụng hàm close() để thực hiện đóng tập tin khi không còn thao tác với nó nữa.

Kết quả như sau:

```
$ ./showfile.py
Enter the filename: sample.txt
I love Python
Pradeepto loves KDE
Sankarshan loves Openoffice
```

Writing in a file (ghi nội dung lên tập tin)

Chúng ta mở tập tin sau đó ghi một vài thông tin ngẫu nhiên vào nó sử dụng hàm write().

```
>>> fobj = open("ircnicks.txt", 'w')
>>> fobj.write('powerpork\n')
>>> fobj.write('indrag\n')
>>> fobj.write('mishti\n')
>>> fobj.write('sankarshan')
>>> fobj.close()
```

Giờ chúng ta sẽ đọc tập tin chúng ta vừa tạo

```
>>> fobj = open('ircnicks.txt')
>>> s = fobj.read()
>>> print s
powerpork
```

```
indrag
mishti
sankarshan
```

copyfile.py

Trong ví dụ này chúng ta sẽ sao chép tập tin văn bản này sang tập tin khác.

```
#!/usr/bin/env python
import sys
if len(sys.argv) < 3:
    print "Wrong parameter"
    print "./copyfile.py file1 file2"
    sys.exit(1)
f1 = open(sys.argv[1])
s = f1.read()
f1.close()
f2 = open(sys.argv[2], 'w')
f2.write(s)
f2.close()
```

Ghi nhớ

Đây là cách để đọc một tập tin nhưng không phải là một ý tưởng hay, một tập tin với dung lượng rất lớn để đọc và sẽ tốn nhiều bộ nhớ. Có một cách khác là tạo ra một tập tin mới với dung lượng nhỏ hơn rồi thực hiện đọc chúng.

Bạn có thể thấy chúng ta một module mới ở đây là sys. sys.argv chưa tất cả các lệnh về dòng. Ghi nhớ câu lênh cp trong dòng lệnh, sau câu lệnh cp chúng ta sao chép tập tin và nó sẽ thành một tập tin mới.

Các giá trị trong sys.argv là các lệnh đầu tiên của nó.

```
#!/usr/bin/env python
import sys
print "First value", sys.argv[0]
print "All values"
for i, x  in enumerate(sys.argv):
    print i, x
```

Kết quả như sau:

```
$ ./argvtest.py Hi there
First value ./argvtest.py
All values
0 ./argvtest.py
1 Hi
2 there
```

Trong ví dụ trên chúng ta sử dụng một hàm mới enumerate(iterableobject), cái mà sẽ trả về chỉ số và giá trị từ iterable.

Random seeking in a file (tìm kiếm ngẫu nhiên)

Bạn có thể sinh ngẫu nhiên nội dung trong tập tin sử dụng seek(). Nó sẽ có hai chiều offset và whence. Bạn có thể đọc thêm trong help của Python.

seek(...) seek(offset[, whence]) -> None. Di chuyên vị chị của tập tin mới.

Ví dụ

```
>>> fobj = open('/tmp/tempfile', 'w')
>>> fobj.write('0123456789abcdef')
>>> fobj.close()
>>> fobj = open('/tmp/tempfile')
>>> fobj.tell()    #tell us the offset position
0L
>>> fobj.seek(5) # Goto 5th byte
>>> fobj.tell()
5L
>>> fobj.read(1) #Read 1 byte
'5'
>>> fobj.seek(-3, 2) # goto 3rd byte from the end
>>> fobj.read() #Read till the end of the file
'def'
```

Đếm dấu cách, tab và dòng mới trong tập tin.

Chúng ta sẽ viết một chương trình đếm dấu cách, tabs, và dòng trong một tập tin có sẵn.

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

import os
import sys


def parse_file(path):
    """
    Parses the text file in the given path and returns space, tab & new line
    details.

    :arg path: Path of the text file to parse

    :return: A tuple with count of spacaes, tabs and lines.
    """
    fd = open(path)
    i = 0
    spaces = 0
    tabs = 0
    for i,line in enumerate(fd):
        spaces += line.count(' ')
        tabs += line.count('\t')
    #Now close the open file
    fd.close()

    #Return the result as a tuple
    return spaces, tabs, i + 1

def main(path):
    """
    Function which prints counts of spaces, tabs and lines in a file.

    :arg path: Path of the text file to parse
    :return: True if the file exits or False.
    """
    if os.path.exists(path):
        spaces, tabs, lines = parse_file(path)
        print "Spaces %d. tabs %d. lines %d" % (spaces, tabs, lines)
        return True
    else:
        return False


if __name__ == '__main__':
    if len(sys.argv) > 1:
        main(sys.argv[1])
    else:
        sys.exit(-1)
    sys.exit(0)
```

Bạn có thể thấy chúng ta có hai hàm trong chương trình, hàm chính và hàm có tên là parse_file nơi là chúng ta xử lý tập tin và trả về kết quả cho hàm chính. Bằng cách chia nhỏ chức năng giúp chúng ta tổ chức được code và có thể viết test case

dễ dàng hơn.

Sử dụng statement

Trong thực tế chúng ta nên sử dụng statement. Chúng sẽ đảm nhiệm việc đóng tập tin cho bạn.

```
>>> with open('setup.py') as fobj:
...     for line in fobj:
...         print line,
...
```

```
#!/usr/bin/env python
"""Factorial project"""
from setuptools import find_packages, setup

setup(name = 'factorial',
    version = '0.1',
    description = "Factorial module.",
    long_description = "A test module for our book.",
    platforms = ["Linux"],
    author="Kushal Das",
    author_email="kushaldas@gmail.com",
    url="http://pymbook.readthedocs.org/en/latest/",
    license = "http://www.gnu.org/copyleft/gpl.html",
    packages=find_packages()
    )
```

Chúc các bạn thành công!

Exceptions

Trong phần này chúng ta sẽ nghiên cứu về exception trong Python và làm sao đề xử lý trong trong code.

Một số lỗi xảy ra trong quá trình chạy chương trình được gọi là exception. Mỗi một exception đều sinh ra một số thông báo lỗi.

NameError

Khi chúng ta bắt đầu viết code, đây sẽ là hầu hết các exceoption mà các bạn có thể tìm thấy. Ví dụ như chúng ta cố gắng truy cập vào biến và nó chưa được khai báo.

```
>>> print kushal
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'kushal' is not defined
```

Dòng cuối cùng chứa nội dung cụ thể của thông báo lỗi, phần còn lại đưa ra thông báo cụ thể làm thế nào để nó xảy ra (hoặc những gì gây ra ngoại lệ).

TypeError

TypeError là một trong các lỗi phổ biến nhất xuất hiện. Nó thường xuất hiện với các chương trình liên quan đến kiểu dữ liệu. Ví dụ:

```
>>> print 1 + "kushal"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Làm sao để xử lý exceptions?

Chúng ta sử dụng cú pháp try... except để xử lý exception. Cú pháp cơ bản như sau:

```
try:
    statements to be inside try clause
    statement2
    statement3
    ...
except ExceptionName:
    statements to evaluated in case of ExceptionName happens
```

Nó làm theo các bước sau:

```
    First all lines between try and except statements.
    If ExceptionName happens during execution of the statements then except clause statements execute
    If no exception happens then the statements inside except clause does not execute.
    If the Exception is not handled in the except block then it goes out of try block.
```

Ví dụ sau sẽ thực hiện kịch bản trên.

```
>>> def get_number():
```

```
...     "Returns a float number"
...     number = float(raw_input("Enter a float number: "))
...     return number
...
>>>
>>> while True:
...     try:
...         print get_number()
...     except ValueError:
...         print "You entered a wrong value"
...
```

```
Enter a float number: 45.0
45.0
Enter a float number: 24,0
You entered a wrong value
Enter a float number: Traceback (most recent call last):
  File "<stdin>", line 3, in <module>
  File "<stdin>", line 3, in get_number
```

KeyboardInterrupt

Trong lần nhập giá trị đầu tiên chúng ta nhập vào số thực, và sau đó in nó ra màn hình, sau đó chúng ta sẽ đưa vào dấu phẩy trong số thực, và sẽ có lỗi khi chúng ta làm như vậy.

Trong lần thứ 3 chúng ta ấn tổ hợp phím ctrl +C sẽ xuất hiện một lỗi là KeyboardInterrupt, sẽ không có một exception nào.

Sẽ có một cách để tổng hợp lại tất cả các exception đó. Ví dụ các bạn có thể xem dưới đây:

```
>>> try:
...     raw_input() # Press Ctrl+c for a KeyboardInterrupt
... except:
...     print "Unknown Exception"
...
Unknown Exception
```

Raising exceptions

```
>>> raise ValueError("A value error happened.")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: A value error happened.
```

Chúng ta có thể bắt exception như một exception bình thường.

```
>>> try:
...     raise ValueError("A value error happened.")
... except ValueError:
...     print "ValueError in our code."
...
ValueError in our code.
```

Using finally for cleanup

Nếu chúng ta muốn có một vài báo cái mỗi khi thực hiện trong mọi hoàn cảnh, chúng ta có thể sử dụng mệnh đề, nó sẽ chạy trước khi kết thúc biểu thức.

```
>>> try:
```

```
...     fobj = open("hello.txt", "w")
...     res = 12 / 0
... except ZeroDivisionError:
...     print "We have an error in division"
... finally:
...     fobj.close()
...     print "Closing the file object."
...
We have an error in division
Closing the file object.
```

Trong ví dụ này chúng ta đảm bảo chắc chắn rằng tập tinđược mở, đóng. Chúc bạn thành công!

Class

Chương trình đầu tiên.

Để viết một chương trình đầu tiên, bạn cần biết cú pháp của class trong chương trình như sau:

```
class nameoftheclass(parent_class):
    statement1
    statement2
    statement3
```

Trong statement bạn có thể viết bất cứ điều gì, bạn có thể định nghĩa hàm (cái mà chúng ta gọi là method của một class)

```
>>> class MyClass(object):
...     a = 90
...     b = 88
...
>>> p = MyClass()
>>> p
<__main__.MyClass instance at 0xb7c8aa6c>
```

Trong ví dụ trên chúng ta có thể thấy đầu tiên chúng ta sẽ khai báo một class có tên là MyClass, chúng ta sẽ viết một vài biểu thức nào đó trong class đó. Sau khi class được định nghĩa, chúng ta sẽ tạo một object p của class MyClass.

```
>>> dir(p)
['__doc__', '__module__', 'a', 'b']
```

bạn có thể nhìn thấy biến a và b trong nó. **init** method

init is là một method đặc biệt trong class Python, nó là một phương thức để khởi tạo một lớp. Trong ví dụ sau sẽ sử dụng nó.

```
class Student(object):
    """
    Returns a ```Student``` object with the given name, branch and year.

    """
    def __init__(self, name, branch, year):
            self.name = name
            self.branch = branch
            self.year = year
            print "A student object is created"

    def print_details(self):
        """
        Prints the details of the student.
        """
        print "Name:", self.name
        print "Branch:", self.branch
        print "Year:", self.year
```

init is called when ever an object of the class is constructed. That means when ever we will create a student object we will see the message “A student object is created” in the prompt. You can see the first argument to the method is self. It is a special variable which points to the current object (like this in C++). The object is passed implicitly to every method available in it, but we have to get it explicitly in every method while writing the methods. Example shown below.

```
>>> std1 = Student()
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: __init__() takes exactly 4 arguments (1 given)
>>> std1 = Student('Kushal','CSE','2005')
A student object is created
```

In this example at first we tried to create a Student object without passing any argument and Python interpreter complained that it takes exactly 4 arguments but received only one (self). Then we created an object with proper argument values and from the message printed, one can easily understand that **init** method was called as the constructor method.

Now we are going to call print_details() method.

```
>>> std1.print_details()
Name: Kushal
Branch: CSE
Year: 2005
```

Inheritance

In general we human beings always know about inheritance. In programming it is almost the same. When a class inherits another class it inherits all features (like variables and methods) of the parent class. This helps in reusing codes.

In the next example we first create a class called Person and create two sub-classes Student and Teacher. As both of the classes are inherited from Person class they will have all methods of Person and will have new methods and variables for their own purpose. student_teacher.py

```
#!/usr/bin/env python
class Person(object):
    """
    Returns a ```Person``` object with given name.

    """
    def __init__(self, name):
        self.name = name

    def get_details(self):
        "Returns a string containing name of the person"
        return self.name

class Student(Person):
    """
    Returns a ```Student``` object, takes 3 arguments, name, branch, year.

    """
    def __init__(self, name, branch, year):
        Person.__init__(self, name)
        self.branch = branch
        self.year = year

    def get_details(self):
        "Returns a string containing student's details."
        return "%s studies %s and is in %s year." % (self.name, self.branch, self.year)


class Teacher(Person):
    """
    Returns a ```Teacher``` object, takes a list of strings (list of papers) as
    argument.
    """
    def __init__(self, name, papers):
        Person.__init__(self, name)
        self.papers = papers

    def get_details(self):
```

```
        return "%s teaches %s" % (self.name, ','.join(self.papers))
```

```
person1 = Person('Sachin')
student1 = Student('Kushal', 'CSE', 2005)
teacher1 = Teacher('Prashad', ['C', 'C++'])

print person1.get_details()
print student1.get_details()
print teacher1.get_details()
```

Kết quả như sau:

```
$ ./student_teacher.py
Sachin
Kushal studies CSE and is in 2005 year.
Prashad teaches C,C++
```

In this example you can see how we called the **init** method of the class Person in both Student and Teacher classes' **init** method. We also reimplemented get_details() method of Person class in both Student and Teacher class. So, when we are calling get_details() method on the teacher1 object it returns based on the object itself (which is of teacher class) and when we call get_details() on the student1 or person1 object it returns based on get_details() method implemented in it's own class. Multiple Inheritance

One class can inherit more than one classes. It gets access to all methods and variables of the parent classes. The general syntax is:

```
class MyClass(Parentclass1, Parentclass2,...):
    def __init__(self):
        Parentclass1.__init__(self)
        Parentclass2.__init__(self)
        ...
        ...
```

Deleting an object

As we already know how to create an object, now we are going to see how to delete an Python object. We use del for this.

```
>>> s = "I love you"
>>> del s
>>> s
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
NameError: name 's' is not defined
```

del actually decreases reference count by one. When the reference count of an object becomes zero the garbage collector will delete that object. Getters and setters in Python

One simple answer, don't. If you are coming from other languages (read Java), you will be tempted to use getters or setters in all your classes. Please don't. Just use the attributes directly. The following shows a direct example.

```
>>> class Student(object):
...     def __init__(self, name):
...         self.name = name
...
>>> std = Student("Kushal Das")
>>> print std.name
```

```
Kushal Das
>>> std.name = "Python"
>>> print std.name
Python
```

Properties

If you want more fine tuned control over data attribute access, then you can use properties. In the following example of a bank account, we will make sure that no one can set the money value to negative and also a property called inr will give us the INR values of the dollars in the account.

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

class Account(object):
    """The Account class,
    The amount is in dollars.
    """
    def __init__(self, rate):
        self.__amt = 0
        self.rate = rate

    @property
    def amount(self):
        "The amount of money in the account"
        return self.__amt

    @property
    def inr(self):
        "Gives the money in INR value."
        return self.__amt * self.rate

    @amount.setter
    def amount(self, value):
        if value < 0:
            print "Sorry, no negative amount in the account."
            return
        self.__amt = value

if __name__ == '__main__':
    acc = Account(61) # Based on today's value of INR :(
    acc.amount = 20
    print "Dollar amount:", acc.amount
    print "In INR:", acc.inr
    acc.amount = -100
    print "Dollar amount:", acc.amount
```

Kết quả như sau:

```
$ python property.py
Dollar amount: 20
In INR: 1220
Sorry, no negative amount in the account.
Dollar amount: 20
```