25. Kĩ thuật indexing và slicing trong numpy array

Indexing cũng giống với các kiểu dữ liệu khác trong python, như tuple và list.

```
import numpy as np
F = np.array([1, 1, 2, 3, 5, 8, 13, 21])
print "the first element of F, i.e. the element with the index 0"
print "F[0] = ", F[0]
print "the last element of F"
print "F[-1] = ", F[-1]
print "Indexing multidimensional arrays"
A = np.array([ [3.4, 8.7, 9.9],
               [1.1, -7.8, -0.7],
               [4.1, 12.3, 4.8]])
print "A[1][0] = ", A[1][0]
print "Or"
print "A[1, 0] = ", A[1,0]

Output:

the first element of F, i.e. the element with the index 0

F[0] =  1

the last element of F

F[-1] =  21

Indexing multidimensional arrays

A[1][0] =  1.1

Or

A[1, 0] =  1.1
```

Kĩ thuật **slicing** giống với slicing trong list với tuple, và hơn thế nữa nó còn có thể áp dụng với mảng nhiều chiều.

Cú pháp thông thường cho mảng một chiều như sau:

```
A[start:stop:step]
```

```
import numpy as np

S = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

  print "S[2:5] = ",S[2:5]

  print "S[:4] = ", S[:4]

  print "S[6:] = ", S[6:]

  print "S[:] = ", S[:]


Output:



S[2:5] =  [2 3 4]

S[:4] =  [0 1 2 3]
```

```
S[6:] =  [6 7 8 9]

S[:] =  [0 1 2 3 4 5 6 7 8 9]
```

Đối với mảng nhiều chiều, cú pháp sẽ là

A[start:stop:step, start:stop:step, start:stop:step]

, dấu phẩy chính là ngăn cách định nghĩa range của phép slicing.

Ví dụ, A là mảng 5x5 và ý nghĩa “

0:3” trong

A[:3,2:] là range cho hàng, từ hàng 0 đến hàng 3 còn “2:” là range cho cột, từ cột 2 đến cột cuối cùng.

```
import numpy as np

A = np.array([[11,12,13,14,15],

            [21,22,23,24,25],

            [31,32,33,34,35],

            [41,42,43,44,45],

            [51,52,53,54,55]])

  print "A[:3,2:] = ", A[:3,2:]


Output:

A[:3,2:] =  [[13 14 15]

 [23 24 25]

 [33 34 35]]
```

Câu hỏi thực hành:

Làm sao để có được kết quả của toàn bộ 2 hàng cuối cùng của mảng phía trên? Đó là A[-2:,:] hoặc A[3:,:].

Làm thế nào để có được kết quả của những**cột** có chỉ số **lẻ**? Câu trả lời là A[:,1::2]

Như ta đã học ở những bài trước đó, chia sẻ memory giữa các đối tượng là một chủ đề thú vị.

A và A[:,1::2] bằng cách nào đó mà python hỗ trợ việc chia sẻ memory này. ta có thể kiểm tra hai đối tượng A, B có chia sẻ memory hay không thông qua phương thức np.may_share_memory(). Điều đó có nghĩa nếu bạn thay đổi giá trị trong B thì A cũng sẽ bị thay đổi.

```
import numpy as np

A = np.array([[11,12,13,14,15],

            [21,22,23,24,25],
```

```
            [31,32,33,34,35],
            [41,42,43,44,45],
            [51,52,53,54,55]])
B = A[:,1::2]
  print "are A and B sharing memory? ", np.may_share_memory(A, B) == True
print "modifing B to see the change in A"
  B[0][0] = 1000
  print "are A and B still sharing memory? ", np.may_share_memory(A, B) == True
print "A = ", A
  print "B = ", B

Output:


are A and B sharing memory?  True
modifing B to see the change in A
are A and B still sharing memory?  True
A =  [[  11 1000   13   14   15]
 [  21   22   23   24   25]
 [  31   32   33   34   35]
 [  41   42   43   44   45]
 [  51   52   53   54   55]]
B =  [[1000   14]
 [  22   24]
 [  32   34]
 [  42   44]
 [  52   54]]
```

Hàm reshape()

Một tiện ích khác với numpy đó là hàm reshape nó sẽ trả về một mảng mới với shape mới tương tự với phép gán x.shape = (tuple).

Mặc dù A và B share memory tuy nhiên chúng là hai đối tượng array khác nhau nên nếu ta kiểm tra id(A) và id(B) ta sẽ thấy chúng khác nhau.

```
import numpy as np
A = np.array([[11,12,13,14,15],
            [21,22,23,24,25],
```

```
            [31,32,33,34,35],
            [41,42,43,44,45],
            [51,52,53,54,55]])
A = np.arange(12)
B = A.reshape(3, 4)
A[0] = 42
  print "B = ",B
  if id(A) != id(B) and np.may_share_memory(A, B):
     print "A and B share same memory, but they are two different objects"

Output:
B =  [[42  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
A and B share same memory, but they are two different objects
```

Kết luận

Về cơ bản kĩ thuật indexing và slicing trong numpy array cũng tương tự như các kiểu dữ liệu list hoặc tuple trong Python cơ bản. Nhưng nó được mở rộng hơn và áp dụng được với các mảng nhiều chiều. Các đối tượng numpy array khác nhau có khả năng chia sẻ memory và khi một đối tượng thay đổi giá trị thì đối tượng còn lại cũng thay đổi theo. Chúng ta có thể đổi chiều của numpy array thông qua hàm reshape().