NumPy - Advanced Indexing

It is possible to make a selection from ndarray that is a non-tuple sequence, ndarray object of integer or Boolean data type, or a tuple with at least one item being a sequence object. Advanced indexing always returns a copy of the data. As against this, the slicing only presents a view.

There are two types of advanced indexing − **Integer** and **Boolean**.

Integer Indexing

This mechanism helps in selecting any arbitrary item in an array based on its Ndimensional index. Each integer array represents the number of indexes into that dimension. When the index consists of as many integer arrays as the dimensions of the target ndarray, it becomes straightforward.

In the following example, one element of specified column from each row of ndarray object is selected. Hence, the row index contains all row numbers, and the column index specifies the element to be selected.

Example 1

Live Demo

```
import numpy as np


x = np.array([[1, 2], [3, 4], [5, 6]])

y = x[[0,1,2], [0,1,0]]

print y
```

Its output would be as follows −

```
[1  4  5]
```

The selection includes elements at (0,0), (1,1) and (2,0) from the first array.

In the following example, elements placed at corners of a 4X3 array are selected. The row indices of selection are [0, 0] and [3,3] whereas the column indices are [0,2] and [0,2].

Example 2

Live Demo

```
import numpy as np

x = np.array([[ 0,  1,  2],[ 3,  4,  5],[ 6,  7,  8],[ 9, 10, 11]])


print 'Our array is:'

print x

print '\n'


rows = np.array([[0,0],[3,3]])

cols = np.array([[0,2],[0,2]])

y = x[rows,cols]


print 'The corner elements of this array are:'

print y
```

The output of this program is as follows −

```
Our array is:
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]

The corner elements of this array are:
[[ 0  2]
 [ 9 11]]
```

The resultant selection is an ndarray object containing corner elements.

Advanced and basic indexing can be combined by using one slice (:) or ellipsis (…) with an index array. The following example uses slice for row and advanced index for column. The result is the same when slice is used for both. But advanced index results in copy and may have different memory layout.

Example 3

Live Demo

```
import numpy as np

x = np.array([[ 0,  1,  2],[ 3,  4,  5],[ 6,  7,  8],[ 9, 10, 11]])


print 'Our array is:'

print x

print '\n'


# slicing

z = x[1:4,1:3]


print 'After slicing, our array becomes:'

print z

print '\n'


# using advanced index for column

y = x[1:4,[1,2]]


print 'Slicing using advanced index for column:'

print y
```

The output of this program would be as follows −

```
Our array is:
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]

After slicing, our array becomes:
[[ 4  5]
 [ 7  8]
 [10 11]]
```

```
Slicing using advanced index for column:
[[ 4  5]
 [ 7  8]
 [10 11]]
```

Boolean Array Indexing

This type of advanced indexing is used when the resultant object is meant to be the result of Boolean operations, such as comparison operators.

Example 1

In this example, items greater than 5 are returned as a result of Boolean indexing.

Live Demo

```
import numpy as np

x = np.array([[ 0,  1,  2],[ 3,  4,  5],[ 6,  7,  8],[ 9, 10, 11]])


print 'Our array is:'

print x

print '\n'


# Now we will print the items greater than 5

print 'The items greater than 5 are:'

print x[x > 5]
```

The output of this program would be –

```
Our array is:
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]

The items greater than 5 are:
[ 6  7  8  9 10 11]
```

Example 2

In this example, NaN (Not a Number) elements are omitted by using ~ (complement operator).

Live Demo

```
import numpy as np

a = np.array([np.nan, 1,2,np.nan,3,4,5])

print a[~np.isnan(a)]
```

Its output would be −

```
[ 1.   2.   3.   4.   5.]
```

Example 3

The following example shows how to filter out the non-complex elements from an array.

Live Demo

```
import numpy as np

a = np.array([1, 2+6j, 5, 3.5+5j])

print a[np.iscomplex(a)]
```

Here, the output is as follows −

```
[2.0+6.j  3.5+5.j]
```