# Fancy Indexing

In the previous sections, we saw how to access and modify portions of arrays using simple indices (e.g., arr[0]), slices (e.g., arr[:5]), and Boolean masks (e.g., arr[arr > 0]). In this section, we'll look at another style of array indexing, known as *fancy indexing*. Fancy indexing is like the simple indexing we've already seen, but we pass arrays of indices in place of single scalars. This allows us to very quickly access and modify complicated subsets of an array's values.

## Exploring Fancy Indexing

Fancy indexing is conceptually simple: it means passing an array of indices to access multiple array elements at once. For example, consider the following array:

import numpy as np  
rand = np.random.RandomState(42)  
  
x = rand.randint(100, size=10)  
print(x)

[51 92 14 71 60 20 82 86 74 74]

Suppose we want to access three different elements. We could do it like this:

[x[3], x[7], x[2]]

[71, 86, 14]

Alternatively, we can pass a single list or array of indices to obtain the same result:

ind = [3, 7, 4]  
x[ind]

array([71, 86, 60])

When using fancy indexing, the shape of the result reflects the shape of the *index arrays* rather than the shape of the *array being indexed*:

ind = np.array([[3, 7],  
 [4, 5]])  
x[ind]

array([[71, 86],  
 [60, 20]])

Fancy indexing also works in multiple dimensions. Consider the following array:

X = np.arange(12).reshape((3, 4))  
X

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

Like with standard indexing, the first index refers to the row, and the second to the column:

row = np.array([0, 1, 2])  
col = np.array([2, 1, 3])  
X[row, col]

array([ 2, 5, 11])

Notice that the first value in the result is X[0, 2], the second is X[1, 1], and the third is X[2, 3]. So, for example, if we combine a column vector and a row vector within the indices, we get a two-dimensional result:

X[row[:, np.newaxis], col]

array([[ 2, 1, 3],  
 [ 6, 5, 7],  
 [10, 9, 11]])

col

array([2, 1, 3])

X[:, np.newaxis]

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

row[:, np.newaxis]

array([[0],  
 [1],  
 [2]])

Here, each row value is matched with each column vector, exactly as we saw in broadcasting of arithmetic operations. For example:

row[:, np.newaxis] \* col

array([[0, 0, 0],  
 [2, 1, 3],  
 [4, 2, 6]])

It is always important to remember with fancy indexing that the return value reflects the *broadcasted shape of the indices*, rather than the shape of the array being indexed.

## Combined Indexing

For even more powerful operations, fancy indexing can be combined with the other indexing schemes we've seen:

print(X)

[[ 0 1 2 3]  
 [ 4 5 6 7]  
 [ 8 9 10 11]]

We can combine fancy and simple indices:

X[2, [2, 0, 1]]

array([10, 8, 9])

We can also combine fancy indexing with slicing:

X[1:, [2, 0, 1]]

array([[ 6, 4, 5],  
 [10, 8, 9]])

X[:, np.newaxis]

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

And we can combine fancy indexing with masking:

mask = np.array([1, 0, 1, 0], dtype=bool)  
X[row[:, np.newaxis], mask]

array([[ 0, 2],  
 [ 4, 6],  
 [ 8, 10]])

All of these indexing options combined lead to a very flexible set of operations for accessing and modifying array values.

## Modifying Values with Fancy Indexing

Just as fancy indexing can be used to access parts of an array, it can also be used to modify parts of an array. For example, imagine we have an array of indices and we'd like to set the corresponding items in an array to some value:

x = np.arange(10)  
i = np.array([2, 1, 8, 4])  
x[i] = 99  
print(x)

[ 0 99 99 3 99 5 6 7 99 9]

We can use any assignment-type operator for this. For example:

x[i] -= 10  
print(x)

[ 0 89 89 3 89 5 6 7 89 9]

Notice, though, that repeated indices with these operations can cause some potentially unexpected results. Consider the following:

x = np.zeros(10)  
x[[0, 0]] = [4, 6]  
print(x)

[ 6. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

Where did the 4 go? The result of this operation is to first assign x[0] = 4, followed by x[0] = 6. The result, of course, is that x[0] contains the value 6.