```
# generate 9 elements evenly spaced between 0 and 5
a = np.linspace(0,5,9)
a
'Output': array([ 0.   ,  0.625,  1.25 ,  1.875,  2.5  ,  3.125,  3.75 ,
4.375,  5.   ])
# the original shape
a.shape
'Output': (9,)
# call the reshape method
a.reshape(3,3)
'Output':
array([[ 0.   ,  0.625,  1.25 ],
       [ 1.875,  2.5  ,  3.125],
       [ 3.75 ,  4.375,  5.   ]])
# the original array maintained its shape
a.shape
'Output': (9,)
# call the resize method - resize does not return an array
a.resize(3,3)
# the resize method has changed the shape of the original array
a.shape
'Output': (3, 3)
```

Stacking Arrays

NumPy has methods for concatenating arrays – also called stacking. The methods hstack and vstack are used to stack several arrays along the horizontal and vertical axis, respectively.

```
# create a 2x2 matrix of random integers in the range of 1 to 20
A = np.random.randint(1, 50, size=[3,3])
B = np.random.randint(1, 50, size=[3,3])
# print out the arrays
A
```

```
'Output':
array([[19, 40, 31],
       [ 5, 16, 38],
       [22, 49,  9]])

B
'Output':
array([[15, 22, 16],
       [49, 26,  9],
       [42, 13, 39]])
```

Let's stack **A** and **B** horizontally using **hstack**. To use **hstack**, the arrays must have the same number of rows. Also, the arrays to be stacked are passed as a tuple to the **hstack** method.

```
# arrays are passed as tuple to hstack
np.hstack((A,B))
'Output':
array([[19, 40, 31, 15, 22, 16],
       [ 5, 16, 38, 49, 26,  9],
       [22, 49,  9, 42, 13, 39]])
```

To stack **A** and **B** vertically using **vstack**, the arrays must have the same number of columns. The arrays to be stacked are also passed as a tuple to the **vstack** method.

```
# arrays are passed as tuple to hstack
np.vstack((A,B))
'Output':
array([[19, 40, 31],
       [ 5, 16, 38],
       [22, 49,  9],
       [15, 22, 16],
       [49, 26,  9],
       [42, 13, 39]])
```

Broadcasting

NumPy has an elegant mechanism for arithmetic operation on arrays with different dimensions or shapes. As an example, when a scalar is added to a vector (or 1-D array). The scalar value is conceptually broadcasted or stretched across the rows of the array and added element-wise. See Figure 10-5.

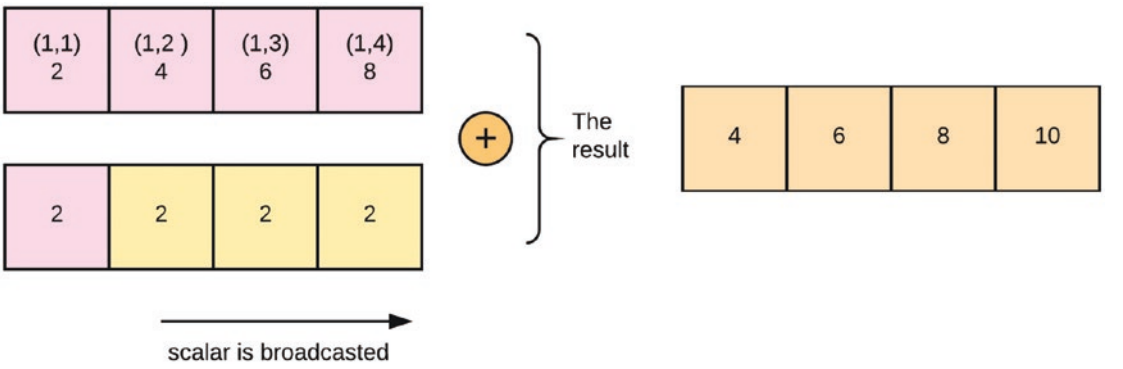

Figure 10-5. *Broadcasting example of adding a scalar to a vector (or 1-D array)*

Matrices with different shapes can be broadcasted to perform arithmetic operations by stretching the dimension of the smaller array. Broadcasting is another vectorized operation for speeding up matrix processing. However, not all arrays with different shapes can be broadcasted. For broadcasting to occur, the trailing axes for the arrays must be the same size or 1.

In the example that follows, the matrices **A** and **B** have the same rows, but the column of matrix **B** is 1. Hence, an arithmetic operation can be performed on them by broadcasting and adding the cells element-wise.

```
A      (2d array):  4 x 3       + <perform addition>
B      (2d array):  4 x 1
Result (2d array):  4 x 3
```

See Figure 10-6 for more illustration.