In [1]:

```
# the following script returns all the rows but only the first two
# columns.

import numpy as np
row1 = [10,12,13]
row2 = [45,32,16]
row3 = [45,32,16]
nums_2d = np.array([row1, row2, row3])
print(nums_2d[:,:2])
```

```
[[10 12]
 [45 32]
 [45 32]]
```

In [2]:

```
# Let’s see another example of slicing. Here, we will slice the rows from row
# one to the end of rows and column one to the end of columns. (Remember,
# row and column numbers start from 0.) In the output, you will see the last
# two rows and the last two columns.

import numpy as np
row1 = [10,12,13]
row2 = [45,32,16]
row3 = [45,32,16]
nums_2d = np.array([row1, row2, row3])
print(nums_2d[1:,1:])
```

```
[[32 16]
 [32 16]]
```

In [3]:

```
# Broadcasting allows you to perform various operations between NumPy
# arrays of different shapes. This is best explained with the help of an example.
# In the script below, you define two arrays: a one-dimensional array of three
# items and another scaler array that contains only one item. Next, the two
# arrays are added. Finally, in the output, you will see that the scaler value, i.e.,
# 10 is added to all the items in the array that contains three items. This is an
# amazing ability and is extremely useful in many scenarios, such as machine
# learning and artificial neural networks.

import numpy as np
array1 = np.array ([14,25,31])
array2 = np.array([10])
result = array1 + array2
print(result)
```

```
[24 35 41]
```

In [5]:

```
# Let’s see another example of broadcasting. In the script below, you add a
# two-dimensional array with three rows and four columns to a scaler array
# with one item. In the output, you will see that the scaler value, i.e., 10, will be
# added to all the 12 items in the first array, which contains three rows and four
# columns.

import numpy as np
array1 = np.random.randint(1,20, size = (3,4))
print(array1)
array2 = np.array([10])
print("after broadcasting")
result = array1 + array2
print(result)
```

```
[[13  3 14 19]
 [17 13  3 18]
 [ 1  9  7 14]]
after broadcasting
[[23 13 24 29]
 [27 23 13 28]
 [11 19 17 24]]
```

In [6]:

```
# You can also use broadcasting to perform operations between twodimensional
# and one-dimensional arrays.
# For example, the following script creates two arrays: a two-dimensional array
# of three rows and four columns and a one-dimensional array of one row and
# four columns. If you perform addition between these two rows, you will see
# that the values in a one-dimensional array will be added to the corresponding
# columns’ values in all the rows in the two-dimensional array.
# For instance, the first row in the two-dimensional array contains values 4, 6,
# 1, and 2. When you add the one-dimensional array with values 5, 10, 20, and
# 25 to it, the first row of the two-dimensional array becomes 9, 16, 21, and 27.

import numpy as np
array1 = np.random.randint(1,20, size = (3,4))
print(array1)
array2 = np.array([5,10,20,25])
print("after broadcasting")
result = array1 + array2
print(result)
```

```
[[ 4 12 18 13]
 [ 1  9 11  1]
 [ 7  1  1 16]]
after broadcasting
[[ 9 22 38 38]
 [ 6 19 31 26]
 [12 11 21 41]]
```

In [8]:

```
# Finally, let’s see an example where an array with three rows and one column
# is added to another array with three rows and four columns. Since, in this
# case, the values of row match, therefore, in the output, for each column in the
# two-dimensional array, the values from the one- dimensional array are added
# row-wise. For instance, the first column in the two-dimensional array
# contains the values 10, 19, and 11. When you add the one-dimensional array
# (5, 10, 20) to it, the new column value becomes 15, 29, and 31.

import numpy as np
array1 = np.random.randint(1,20, size = (3,4))
print(array1)
array2 = np.array([[5],[10],[20]])
print("after broadcasting")
result = array1 + array2
print(result)
```

```
[[ 2  6 13 14]
 [11  1  6  2]
 [10 12  5  5]]
after broadcasting
[[ 7 11 18 19]
 [21 11 16 12]
 [30 32 25 25]]
```

In [9]:

```
# There are two main ways to copy an array in NumPy. You can either copy
# the contents of the original array, or you can copy the reference to the
# original array into another array.
# To copy the contents of the original array into a new array, you can call the
# copy() function on the original array. Now, if you modify the contents of the
# new array, the contents of the original array are not modified.
# For instance, in the script below, in the copied array2, the item at index 1 is
# updated. However, when you print the original array, you see that the index
# one for the original array is not modified.

import numpy as np
array1 = np.array([10,12,14,16,18,20])
array2 = array1.copy()
array2[1] = 20
print(array1)
print(array2)
```

```
[10 12 14 16 18 20]
[10 20 14 16 18 20]
```

In [10]:

```
# The other method to copy an array in Python is via the view() method.
# However, with the view method, if the contents of a new array are modified,
# the original array is also modified. Here is an example:

import numpy as np
array1 = np.array([10,12,14,16,18,20])
array2 = array1.view()
array2[1] = 20
print(array1)
print(array2)
```

```
[10 20 14 16 18 20]
[10 20 14 16 18 20]
```

In [12]:

```
# You can save and load NumPy arrays to and from your local drive.
# To save a NumPy array, you need to call the save() method from the NumPy
# module and pass it the file name for your NumPy as the first argument, while
# the array object itself as the second argument. Here is an example:

import numpy as np
array1 = np.array([10,12,14,16,18,20])
np.save("D:\internship 2", array1)
```

In [13]:

```
# The save() method saves a NumPy array in “NPY” file format. You can also
# save a NumPy array in the form of a text file, as shown in the following
# script:

import numpy as np
array1 = np.array([10,12,14,16,18,20])
np.savetxt("D:\internship 2\my_array.txt", array1)
```