In [1]:

```
# If you want to delete multiple items from an array, you can pass the item
# indexes in the form of a list to the delete() method. For example, the
# following script deletes the items at index 1 and 2 from the NumPy array
# named my_array.

import numpy as np
my_array = np.array(["Red", "Green", "Orange"])
print(my_array)
print("After deletion")
updated_array = np.delete(my_array, [1,2])
print(updated_array)
```

```
['Red' 'Green' 'Orange']
After deletion
['Red']
```

In [2]:

```
# You can delete a row or column from a 2-D array using the delete method.
# However, just as you did with the append() method for adding items, you
# need to specify whether you want to delete a row or column using the axis
# attribute.
# The following script creates an integer array with four rows and five
# columns. Next, the delete() method is used to delete the row at index 1
# (second row). Notice here that to delete the array, the value of the axis
# attribute is set to 0.

import numpy as np
integer_random = np.random.randint(1,11, size=(4, 5))
print(integer_random)
print("After deletion")
updated_array = np.delete(integer_random, 1, axis = 0)
print(updated_array)
```

```
[[ 3  1  8  5  7]
 [ 2  2  7  4 10]
 [ 4  6  6  3  8]
 [ 5  8  6  8  7]]
After deletion
[[3 1 8 5 7]
 [4 6 6 3 8]
 [5 8 6 8 7]]
```

In [3]:

```
# Finally, to delete a column, you can set the value of the axis attribute to 1, as
# shown below:

import numpy as np
integer_random = np.random.randint(1,11, size=(4, 5))
print(integer_random)
print("After deletion")
updated_array = np.delete(integer_random, 1, axis = 1)
print(updated_array)
```

```
[[ 4  7 10  3 10]
 [ 8  5  4  5 10]
 [ 8 10  2  3  5]
 [ 9  1  7  5  5]]
After deletion
[[ 4 10  3 10]
 [ 8  4  5 10]
 [ 8  2  3  5]
 [ 9  7  5  5]]
```

In [4]:

```
# You can sort NumPy arrays of various types. Numeric arrays are sorted by
# default in ascending order of numbers. On the other hand, text arrays are
# sorted alphabetically.
# To sort an array in NumPy, you may call the np.sort() function and pass it to
# your NumPy array. The following script shows how to sort a NumPy array of
# 10 random integers between 1 and 20.

import numpy as np
print("unsorted array")
my_array = np.random.randint(1,20,10)
print(my_array)
print("\nsorted array")
sorted_array = np.sort(my_array)
print(sorted_array)
```

```
unsorted array
[19 19  4  7 16 18  2 18  8 17]

sorted array
[ 2  4  7  8 16 17 18 18 19 19]
```

In [6]:

```
# As mentioned earlier, text arrays are sorted in alphabetical order. Here is an
# example of how you can sort a text array with the NumPy sort() method.

import numpy as np
print("unsorted array")
my_array = np.array(["Red", "Green", "Blue", "Yellow"])
print(my_array)
print("\nsorted array")
sorted_array = np.sort(my_array)
print(sorted_array)
```

```
unsorted array
['Red' 'Green' 'Blue' 'Yellow']

sorted array
['Blue' 'Green' 'Red' 'Yellow']
```

In [7]:

```
# Finally, Boolean arrays are sorted in a way that all the False values appear
# first in an array. Here is an example of how you can sort the Boolean arrays
# in NumPy.

import numpy as np
print("unsorted array")
my_array = np.array([False, True, True, False, False, True, False, True])
print(my_array)
print("\nSorted array")
sorted_array = np.sort(my_array)
print(sorted_array)
```

```
unsorted array
[False  True  True False False  True False  True]

Sorted array
[False False False False  True  True  True  True]
```

In [9]:

```
# NumPy also allows you to sort two-dimensional arrays. In two-dimensional
# arrays, each item itself is an array. The sort() function sorts an item in each
# individual array in a two-dimensional array.
# The script below creates a two-dimensional array of shape (4,6) containing
# random integers between 1 to 20. The array is then sorted via the np.sort()
# method.

import numpy as np
print("unsorted array")
my_array = np.random.randint(1,20, size = (4,6))
print(my_array)
print("Sorted array")
sorted_array = np.sort(my_array)
print(sorted_array)
```

```
unsorted array
[[ 9 16  3 10  7  4]
 [ 2 18 10  9  8 10]
 [ 8 13  1  8 10 11]
 [ 9  3  6  4 13 10]]
Sorted array
[[ 3  4  7  9 10 16]
 [ 2  8  9 10 10 18]
 [ 1  8  8 10 11 13]
 [ 3  4  6  9 10 13]]
```

In [10]:

```
# You can also sort an array in descending order. To do so, you can first sort an
# array in ascending order via the sort() method. Next, you can pass the sorted
# array to the flipud() method, which reverses the sorted array and returns the
# array sorted in descending order. Here is an example of how you can sort an
# array in descending order.

import numpy as np
print("unsorted array")
my_array = np.random.randint(1,20,10)
print(my_array)
print("\nsorted array")
sorted_array = np.sort(my_array)
reverse_sorted = np.flipud(sorted_array)
print(reverse_sorted)
```

```
unsorted array
[ 5 17 18 16  4 12  9 16  1  4]

sorted array
[18 17 16 16 12  9  5  4  4  1]
```

In [11]:

```
# You can also modify the shape of a NumPy array. To do so, you can use the
# reshape() method and pass it the new shape for your NumPy array.
# In this section, you will see how to reshape a NumPy array from lower to
# higher dimensions and vice versa.

# The following script defines a one-dimensional array of 10 random integers
# between 1 and 20. The reshape() function reshapes the array into the shape
# (2,5).

import numpy as np
print("one-dimensional array")
one_d_array = np.random.randint(1,20,10)
print(one_d_array)
print("\ntwo-dimensional array")
two_d_array = one_d_array.reshape(2,5)
print(two_d_array)
```

```
one-dimensional array
[12  9 14  8 18 12  3 10 11  5]

two-dimensional array
[[12  9 14  8 18]
 [12  3 10 11  5]]
```

In [12]:

```
# It is important to mention that the product of the rows and columns of the
# original array must match the value of the product of rows and columns of
# the reshaped array. For instance, the shape of the original array in the last
# script was (10,) with product 10. The product of the rows and columns in the
# reshaped array was also 10 (2 x 5)
# You can also call the reshape() function directly from the NumPy module and
# pass it the array to be reshaped as the first argument and the shape tuple as
# the second argument. Here is an example which converts an array of shape
# (10,) to (2,5).

import numpy as np
print("one-dimensional array")
one_d_array = np.random.randint(1,20,10)
print(one_d_array)
print("\ntwo-dimensional array")
two_d_array = np.reshape(one_d_array,(2,5))
print(two_d_array)
```

```
one-dimensional array
[13  1 13  3  5 11  2 10 14  1]

two-dimensional array
[[13  1 13  3  5]
 [11  2 10 14  1]]
```