Basics of NumPy Arrays - 2

In [2]:

```
import numpy as np
```

Array Indexing

Similar to list indexing, arrays can be accessed using the indices.

In [3]:

```
nparray = np.array([1, 2, 3, 4, 5])
```

In [5]:

```
#Access first element, indexing starts from 0
nparray[0]
```

Out[5]:

```
1
```

In [6]:

```
#Access second element, indexing starts from 0
nparray[1]
```

Out[6]:

```
2
```

In [7]:

```
#Access last element, negative indices works
nparray[-1]
```

Out[7]:

```
5
```

In [8]:

```
nparray[-2]
```

Out[8]:

```
4
```

In multi-dimensional arrays, comma separated tupe of indices can be used to access elements.

In [12]:

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

Out[12]:

```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

In [10]:

```
#Access element at location (0, 0), zeroth row and zeroth column
x[0,0]
```

Out[10]:

```
1
```

In [11]:

```
#Access element at location (1, 2), first row and second column
x[1,2]
```

Out[11]:

```
6
```

Values can be modified using indices,

In [13]:

```
#Modify the (0, 0)th entry to 555
x[0, 0] = 555
x
```

Out[13]:

```
array([[555,   2,   3],
       [  4,   5,   6],
       [  7,   8,   9]])
```

Array Slicing

[] can be used access the sub arrays i.e. array sicing is supported similar to list. To access a slice of array x , use :
x[start:stop:step]

In [40]:

```
nparray
```

Out[40]:

```
array([1, 2, 3, 4, 5])
```

In [43]:

```
nparray[1:]  # all values after index 1
```

Out[43]:

```
array([2, 3, 4, 5])
```

In [48]:

```
nparray[ : 3] # all values till index 3
```

Out[48]:

```
array([1, 2, 3])
```

In [49]:

```
nparray[2:4] # all value between index 2 and 4
```

Out[49]:

```
array([3, 4])
```

In [47]:

```
nparray[::2]  #every other element
```

Out[47]:

```
array([1, 3, 5])
```

In [51]:

```
nparray[1::2]  #every other element starting from element at index 1
```

Out[51]:

```
array([2, 4])
```

In [52]:

```
nparray[::-1] # all elements in reversed direction
```

Out[52]:

```
array([5, 4, 3, 2, 1])
```

In [53]:

```
x
```

Out[53]:

```
array([[555,   2,   3],
       [  4,   5,   6],
       [  7,   8,   9]])
```

In [55]:

```
x[ : 2, : 3] # first two rows, first three columns
```

Out[55]:

```
array([[555,   2,   3],
       [  4,   5,   6]])
```

In [57]:

```
#Accessing array rows
x[0, :]  # zeroth row , all columns
```

Out[57]:

```
array([555,   2,   3])
```

In [59]:

```
#Accessing array columns
x[:, 1]  # all row , second columns
```

Out[59]:

```
array([2, 5, 8])
```

Array copying

Array slices are views, hence if they are modified then original array is changed.

In [60]:

```
nparray
```

Out[60]:

```
array([1, 2, 3, 4, 5])
```

In [63]:

```
nparray_sub = nparray[2:4]   #Slice the array
nparray_sub
```

Out[63]:

```
array([3, 4])
```

In [64]:

```
nparray_sub[0] = 5  # modify the element of slice
nparray_sub
```

Out[64]:

```
array([5, 4])
```

In [66]:

```
nparray #original array is changed
```

Out[66]:

```
array([1, 2, 5, 4, 5])
```

In order to prevent this behaviour copy needs to be done.

In [70]:

```
nparray_copy = nparray[2:4].copy()  #create copy of array slice
```

In [74]:

```
nparray_copy[0] = 90  # modify the slice
nparray_copy
```

Out[74]:

```
array([90,  4])
```

In [76]:

```
nparray  #original array is still intact
```

Out[76]:

```
array([1, 2, 5, 4, 5])
```

Reshaping of Arrays

Reshape can be used to change the structure of array. Size of original and reshaped arrays should be same.

In [80]:

```
a = [ 1, 2, 3, 4, 5, 6, 7, 8, 9]
array = np.array(a)
array
```

Out[80]:

```
array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [81]:

```
array.reshape(3, 3)
```

Out[81]:

```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

Reshaping can be used to convert an array into row matrix or column matrix.

In [77]:

```
x
```

Out[77]:

```
array([[555,   2,   3],
       [  4,   5,   6],
       [  7,   8,   9]])
```

In [78]:

```
x.reshape(9, 1)
```

Out[78]:

```
array([[555],
       [  2],
       [  3],
       [  4],
       [  5],
       [  6],
       [  7],
       [  8],
       [  9]])
```

In [79]:

```
x.reshape(1, 9)
```

Out[79]:

```
array([[555,   2,   3,   4,   5,   6,   7,   8,   9]])
```

Array Concatenation

Joining of two arrays is possible through np.concatenate, np.vstack and np.hstack.

In [84]:

```
x = np.array([1, 2, 3])
y = np.array([10, 20, 30])
z = np.array([99, 99, 99])
```

In [86]:

```
np.concatenate([x, y, z])
```

Out[86]:

```
array([ 1,  2,  3, 10, 20, 30, 99, 99, 99])
```

In [89]:

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

Out[89]:

```
array([[1, 2, 3],
       [4, 5, 6]])
```

In [90]:

```
np.concatenate([two_d_array, two_d_array])
```

Out[90]:

```
array([[1, 2, 3],
       [4, 5, 6],
       [1, 2, 3],
       [4, 5, 6]])
```

In [92]:

```
np.vstack([two_d_array, y])
```

Out[92]:

```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [10, 20, 30]])
```

In [95]:

```
v_array = np.array([[99], [99]])
```

In [97]:

```
np.hstack([two_d_array, v_array])
```

Out[97]:

```
array([[ 1,  2,  3, 99],
       [ 4,  5,  6, 99]])
```

Array Splitting

Splititng is done by split, vsplit and hsplit.

In [98]:

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

In [101]:

```
n1, n2, n3, n4 = np.split(x, [2, 5, 8 ])   # splitting position 2, 5, and 8
```

In [102]:

```
print(n1, n2, n3, n4)
```

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

In [104]:

```
grid = np.arange(5, 20).reshape(5,3)
grid
```

Out[104]:

```
array([[ 5,  6,  7],
       [ 8,  9, 10],
       [11, 12, 13],
       [14, 15, 16],
       [17, 18, 19]])
```

In [110]:

```
n1, n2 = np.split(grid, [3])  #splitting postion third row
```

In [109]:

```
print(n1, "\n\n", n2)
```

```
[[ 5  6  7]
 [ 8  9 10]
 [11 12 13]]

 [[14 15 16]
 [17 18 19]]
```

In [112]:

```
n1, n2 = np.vsplit(grid, [1])
```

In [113]:

```
print(n1, "\n\n", n2)
```

```
[[5 6 7]]

 [[ 8  9 10]
 [11 12 13]
 [14 15 16]
 [17 18 19]]
```

In [114]:

```
n1, n2 = np.hsplit(grid, [1])
```

In [115]:

```
print(n1, "\n\n", n2)
```

```
[[ 5]
 [ 8]
 [11]
 [14]
 [17]]

 [[ 6  7]
 [ 9 10]
 [12 13]
 [15 16]
 [18 19]]
```

Fancy Indexing

Here arrays of indices is passed to access the elements.

In [16]:

```
nparray
```

Out[16]:

```
array([1, 2, 3, 4, 5])
```

In [21]:

```
indices = [1, 3, 4]  #indices first, third and fourth
nparray[indices] #elements at first , third and forth elements are returned
```

Out[21]:

```
array([2, 4, 5])
```

In [38]:

```
#Create a 2-d array using the indices
indices = np.array([[0,1], [3,4]])
nparray[indices]
```

Out[38]:

```
array([[1, 2],
       [4, 5]])
```

In [20]:

```
x
```

Out[20]:

```
array([[555,   2,   3],
       [  4,   5,   6],
       [  7,   8,   9]])
```

In [36]:

```
row = np.array([0, 1])
column = np.array([0,1])
```

In [39]:

```
x[row, column]  #(0,0), (1, 1) are returned
```

Out[39]:

```
array([555,   5])
```