In [1]:
```python
#This is a supplementary material to the lecture "2D Lists and Numpy" to quickly revise, whenever need
ed
```

In [2]:
```python
#2D list
#creating a 2D list
#if we were to create an n sized 1d list with all zeros, then
n = 5
l_1d = [0]*n
#or alternatively
l_1d = [0 for i in range(n)]
print(l_1d)
#2D list can be thought of as collection of multiple 1D lists
#for example, if we want to create a 2D list with say 3 rows and 5 columns with all zeros
#it can be thought of as collection of 3 nos. 1D lists, each containing 5 items
n_rows = 3
n_cols = 5
l_2d = [[0]*n_cols]*n_rows
print(l_2d)
```

```
[0, 0, 0, 0, 0]
[[0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0]]
```

In [3]:
```python
#alternatively, same 2D list can be created as
l_2d = [[0 for j in range(n_cols)] for i in range(n_rows)]
l_2d
```

Out[3]:
```
[[0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0]]
```

In [4]:
```python
#taking 2d list as input from user
#say user gives two lines of input, first line contains #rows and #cols separated by space
#next line contains all the elements of 2D list separated by space
n_rows, n_cols = [int(item) for item in input().strip().split(' ')]
elements  = [int(item) for item in input().strip().split(' ')]
list_2d = [[elements[i * n_cols + j] for j in range(n_cols)] for i in range(n_rows)]
list_2d
```

```
3 4
1 2 3 4 5 6 7 8 9 10 11 12
```

Out[4]:
```
[[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]
```

In [5]:
```python
## accessing an element in 2d list
list_2d[2][1]
```

Out[5]:
```
10
```

In [6]:
```python
#NumPy
#NumPy is the fundamental package for scientific computing with Python.
#to use numpy functions, we need to import the package
import numpy as np
```

In [7]:
```python
#creating numpy array
l = [1, 2, 3, 4]
np_arr = np.array(l)
np_arr
```

Out[7]:
```
array([1, 2, 3, 4])
```

In [8]:
```python
np_arr.dtype
```

Out[8]:
```
dtype('int32')
```

In [9]:
```python
l = [1, 2, '3']           #if some of the items are strings in addition to int, it converts all item
s to string
np_arr = np.array(l)
np_arr
```

Out[9]:
```
array(['1', '2', '3'], dtype='<U11')
```

In [10]:
```python
#to create numpy array of all zeros or ones
arr_zeros = np.zeros(5)       #will create 1d numpy array of size 5 with all zeros
arr_ones = np.ones(5)         #will create 1d numpy array of size 5 with all ones
print(arr_zeros)
print(arr_ones)
```

```
[0. 0. 0. 0. 0.]
[1. 1. 1. 1. 1.]
```

In [11]:
```python
arr_zeros.dtype
```

Out[11]:
```
dtype('float64')
```

In [12]:
```python
#if you want to create array of int type
arr_zeros = np.zeros(5, int)
print(arr_zeros)
arr_zeros.dtype
```

```
[0 0 0 0 0]
```

Out[12]:
```
dtype('int32')
```

In [13]:
```python
#creating multidimensional numpy arrays
arr_2d = np.zeros((3,5))    #will create 2d numpy arrays of 3x5 conatining all zeros
arr_2d
```

Out[13]:
```
array([[0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.]])
```

In [14]:
```python
arr_3d = np.ones((3, 3, 5))       #think of it as 3 nos. of 2d numpy arrays of size 3x5 each containin
g all ones
arr_3d
```

Out[14]:
```
array([[[1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.]],

       [[1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.]],

       [[1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.]]])
```

In [15]:
```python
arr_3d.shape      #to check the shape or dimensions of numpy array
```

Out[15]:
```
(3, 3, 5)
```

In [16]:
```python
#Returns an array containing the same data with a new shape.
arr_2d = arr_3d.reshape(9,5)
```

In [17]:
```python
#accessing the items and slicing work same as we have studied in lists
```

In [18]:
```
#one nice feature is that we can extarct sub-matrix/sub-array of the original matrix/array using slici
ng
arr = np.array([[1,2,3,4], [5,6,7,8], [9, 10, 11, 12]])
arr_sub = arr[1:3, 1:3]         #it will give a 2d array containing items from arr present in rows and
cols with index 1 and 2
arr_sub
```

Out[18]:
```
array([[ 6,  7],
       [10, 11]])
```

In [19]:
```
#numpy operations
arr_new = arr + 2       #will add 2 to each item in arr
print(arr_new)
print(arr_new - arr)    #will do element-wise subtraction of elements
```

```
[[ 3  4  5  6]
 [ 7  8  9 10]
 [11 12 13 14]]
[[2 2 2 2]
 [2 2 2 2]
 [2 2 2 2]]
```

In [20]:
```
print(arr.sum())      #return sum of all elements in arr
print(arr.mean())     #return mean of all elements in arr
```

```
78
6.5
```

In [21]:
```
arr * arr_new       #will do element-wise multiplication of these matrices
```

Out[21]:
```
array([[  3,   8,  15,  24],
       [ 35,  48,  63,  80],
       [ 99, 120, 143, 168]])
```

In [22]:
```
#if, you want to compute the dot product of two numpy arrays, provided dimensions are compatible
a = np.array([[1,2], [3,4]])
b = np.array([[5, 6], [7,8]])
c = a.dot(b)        #will compute the dot product
c
```

Out[22]:
```
array([[19, 22],
       [43, 50]])
```

In [23]:
```
#say, you want to calculate sum of first row of array c
np.sum(c[0,:])
```

Out[23]:
```
41
```

In [24]:
```
#similarly, for sum of elements in second column of c
np.sum(c[:, 1])
```

Out[24]:
```
72
```

In [25]:
```
np.sum(c, axis = 0)       #column-wise sum
```

Out[25]:
```
array([62, 72])
```

In [26]:
```
np.sum(c, axis = 1)       #row-wise sum
```

Out[26]:
```
array([41, 93])
```

In [27]:
```
#Numpy also has lots of ways to create random number arrays:
#Create an array of the given shape and populate it with random samples from a uniform distribution ov
er [0, 1).



np.random.rand(2)
```

Out[27]:
```
array([0.46616062, 0.90643978])
```

In [28]:
```
np.random.rand(5,5)
```

Out[28]:
```
array([[0.46139565, 0.61125049, 0.58092469, 0.96934139, 0.64813732],
       [0.42130718, 0.26109802, 0.30697161, 0.63601977, 0.57425207],
       [0.96214112, 0.57649256, 0.73205837, 0.00952067, 0.80429736],
       [0.42951367, 0.39888115, 0.72890438, 0.4917824 , 0.51476326],
       [0.8908243 , 0.38508019, 0.61038847, 0.40601544, 0.14917933]])
```

In [29]:
```
#Return a sample (or samples) from the "standard normal" distribution. Unlike rand which is uniform:
np.random.randn(2)
```

Out[29]:
```
array([-1.39718995,  1.77372121])
```

In [30]:
```
np.random.randn(5,5)
```

Out[30]:
```
array([[-0.37923369, -0.56998801,  0.3712431 ,  0.78905378,  0.10938529],
       [-0.12640059, -0.15656451,  0.05231141,  0.83405347,  0.68454559],
       [-0.15418843,  0.04013789,  0.61515418,  0.24499148, -0.34176554],
       [-2.4499439 , -1.98456664, -2.26153452,  0.8622766 ,  0.76724156],
       [-0.97318391, -1.33416268,  0.89568369,  0.20433311,  1.00688174]])
```

In [31]:
```
#Return random integers from low (inclusive) to high (exclusive).


np.random.randint(1,100)
```

Out[31]:
```
59
```

In [32]:
```
#Following will randomly sample 10 values in the interval [1, 100)
np.random.randint(1,100,10)
```

Out[32]:
```
array([74,  3, 27, 54, 32, 87, 50, 93,  8, 48])
```

In [33]:
```
#Let's discuss some useful attributes and methods or an array:
arr = np.arange(25)
ranarr = np.random.randint(0,50,10)
```

In [34]:
```
arr
```

Out[34]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24])
```

In [35]:
```
ranarr
```

Out[35]:
```
array([38, 43, 39,  6, 36, 37, 18, 17,  6, 31])
```

In [36]:
```
#Thanks, Happy Coding!
```

In [ ]:
```
#To download .ipynb notebook, right click the following link and click save as
https://ninjasfiles.s3.amazonaws.com/0000000000003219.ipynb
```