Creating 2-D Arrays (Matrices)

Let us construct a simple 2-D array.

```
# construct a 2-D array
my_2D = np.array([[2,4,6],
                  [8,10,12]])
my_2D
'Output':
array([[ 2,  4,  6],
       [ 8, 10, 12]])
# check the number of dimensions
my_2D.ndim
'Output': 2
# get the shape of the 2-D array - this example has 2 rows and
3 columns: (r, c)
my_2D.shape
'Output': (2, 3)
```

Let's explore common methods in practice for creating 2-D NumPy arrays, **which are also matrices**.

```
# create a 3x3 array of ones
np.ones([3,3])
'Output':
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.],
       [ 1.,  1.,  1.]])
# create a 3x3 array of zeros
np.zeros([3,3])
'Output':
array([[ 0.,  0.,  0.],
       [ 0.,  0.,  0.],
       [ 0.,  0.,  0.]])
# create a 3x3 array of a particular scalar - full(shape, fill_value)
np.full([3,3], 2)
```

```
'Output':
array([[2, 2, 2],
       [2, 2, 2],
       [2, 2, 2]])
# create a 3x3, empty uninitialized array
np.empty([3,3])
'Output':
array([[ -2.00000000e+000,  -2.00000000e+000,   2.47032823e-323],
       [  0.00000000e+000,   0.00000000e+000,   0.00000000e+000],
       [ -2.00000000e+000,  -1.73060571e-077,  -2.00000000e+000]])
# create a 4x4 identity matrix - i.e., a matrix with 1's on its diagonal
np.eye(4) # or np.identity(4)
'Output':
array([[ 1.,  0.,  0.,  0.],
       [ 0.,  1.,  0.,  0.],
       [ 0.,  0.,  1.,  0.],
       [ 0.,  0.,  0.,  1.]])
```

Creating 3-D Arrays

Let's construct a basic 3-D array.

```
# construct a 3-D array
my_3D = np.array([[
                    [2,4,6],
                    [8,10,12]
                   ],[
                    [1,2,3],
                    [7,9,11]
                   ]])
my_3D
'Output':
array([[[ 2,  4,  6],
        [ 8, 10, 12]],

       [[ 1,  2,  3],
        [ 7,  9, 11]]])
```