2D Arrays and Matrices

[2]:
```
import numpy as np
```

2D Arrays and Matrices

NumPy arrays can have any number of dimensions, but in this course we will only go up to 2. 2D arrays are quite common if you are working with images or running certain simulations of 3D systems.

You can created 2D arrays from a nested sequence using the `np.array()` function:

[3]:
```
print(
np.array(
    [[1, 22, 45, 6, 3, 2],
      [34, 2, 56, 2, 7, 2],
      [2, 35, 64, 11, 1, 5]]
))
```

```
[[ 1 22 45  6  3  2]
 [34  2 56  2  7  2]
 [ 2 35 64 11  1  5]]
```

When you are doing this, make sure that your dimensions are correct, otherwise you will end up with an array of sequences:

[4]:
```
print(np.array([[1, 2, 3], [4, 5]]))
```

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

Shape and Size

Now would be a good time to talk about the distinction between the `shape` and `size` attributes of an array.

[5]:
```
arr = np.array(
    [[0, 1],
     [0, 1],
     [0, 1]] )
```

The `size` of the array is a count of how many elements the array contains.

[6]:
```
arr.size
```

[6]:
```
6
```

The `shape` of an array is a tuple which tells you the length of each axis:

[7]:
```
arr.shape
```

[7]:
```
(3, 2)
```

Note that **axis 0** (the first value in the tuple) corresponds to the "rows" and **axis 1** (the second value in the tuple) corresponds to the "columns" of the 2D array (this makes more sense when thinking about matrices).

Generating 2D Arrays

You can also generate 2D arrays quickly by using the `np.ones()` and `np.zeros()` functions by specifying the shape the array instead of the size:

[8]:
```
np.ones( (3, 6) )
```

[8]:
```
array([[1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1.]])
```

[9]:
```
np.zeros( (5, 2) )
```

[9]:
```
array([[0., 0.],
       [0., 0.],
       [0., 0.],
       [0., 0.],
       [0., 0.]])
```

Remember that the shape is a **tuple**. It is a common mistake to enter each axis as a separate argument.

Indexing and Slicing

To index a multidimensional array you specify the index you want for each axis:

```
array[axis0_index, axis1_index, axis2_index, ... ]
```

Note the use of commas to separate each axis. For example, let's index the 2D array:

[10]:
```
arr = np.array(
    [[1,  2,  3,  4],
     [5,  6,  7,  8],
     [9, 10, 11, 12]]
)
```

[11]:
```
arr[1, 2]
```

[11]:
```
7
```

[12]:
```
arr[2, -1]
```

[12]:
```
12
```

You can slice multidimensional arrays by separating the slice along each axis by commas:

[13]:
```
arr[:, 1:3]
```

[13]:
```
array([[ 2,  3],
       [ 6,  7],
       [10, 11]])
```

You can extract individual rows and columns by slicing along one axis and indexing the other. For example:

[14]:
```
#Slicing the first row
arr[0, :]
```

[14]:
```
array([1, 2, 3, 4])
```

[18]:
```
#Slicing the third column
arr[:, 2]
```

[18]:
```
array([ 3,  7, 11])
```

[19]:
```
#Slicing the last column
arr[:, -1]
```

[19]:
```
array([ 4,  8, 12])
```

Transpose

You can use the `.T` attribute of an array (or matrix) to get the transpose (swap the rows and columns):

[20]:
```
arr.T
```

[20]:
```
array([[ 1,  5,  9],
       [ 2,  6, 10],
       [ 3,  7, 11],
       [ 4,  8, 12]])
```

Matrices

NumPy's matrices are similar to 2D arrays, except for some matrix specific attributes, methods and operations.

You can create a matrix by using the `np.matrix()` function with a sequence argument:

[21]:
```
np.matrix(
    [[1, 2],
     [3, 4],
     [5, 6]]
)
```

[21]:
```
matrix([[1, 2],
        [3, 4],
        [5, 6]])
```

To generate large, structured matrices, you can use some of the array generating functions:

[22]:
```
np.matrix(np.ones( (2, 3) ))
```

[22]:
```
matrix([[1., 1., 1.],
        [1., 1., 1.]])
```

Slicing and Indexing

Slicing and indexing matrices is the same as for 2D arrays.

Matrix Operations

Consider the 2 by 3 matrix `mat1` and the 3 by 2 matrix `mat2`.

[23]:
```
mat1 = np.matrix(np.ones( (2, 3) ))
mat2 = np.matrix([[1, 2],
                  [3, 4],
                  [5, 6]])
```

Addition, **subtraction** and **division** between matrices are the same as for arrays (vectorized):

[24]:
```
mat1.T + mat2
```

[24]:
```
matrix([[2., 3.],
        [4., 5.],
        [6., 7.]])
```

[25]:
```
mat1.T/mat2
```

[25]:
```
matrix([[1.        , 0.5       ],
        [0.33333333, 0.25      ],
        [0.2       , 0.16666667]])
```

(`mat1` has been transposed to ensure the matrices shape's match)

Multiplication is matrix multiplication:

[26]:
```
mat1*mat2
```

[26]:
```
matrix([[ 9., 12.],
        [ 9., 12.]])
```

[27]:
```
mat2*mat1
```

[27]:
```
matrix([[ 3.,  3.,  3.],
        [ 7.,  7.,  7.],
        [11., 11., 11.]])
```

Inverse

You can use the `.I` attribute to get the multiplicative inverse of a matrix.

[28]:
```
mat = np.matrix(
    [[1, 0, 1],
     [0, 1, 0],
     [1, 0, 2]]
)
```

[29]:
```
mat.I
```

[29]:
```
matrix([[ 2.,  0., -1.],
        [ 0.,  1.,  0.],
        [-1.,  0.,  1.]])
```

[30]:
```
mat*mat.I
```

[30]:
```
matrix([[1., 0., 0.],
        [0., 1., 0.],
        [0., 0., 1.]])
```

[31]:
```
mat.I*mat
```

[31]:
```
matrix([[1., 0., 0.],
        [0., 1., 0.],
        [0., 0., 1.]])
```