Numpy Arrays

Numpy, Short for Numerical Python, is one of the fundamental packages for numerical computing in python.

Numpy can be useful for: 1) ndarray, an efficient multidimensional array providing fast array-oriented arithmetic operations and flexible broadcasting capabilities. 2) Mathematical functions for fast array operations on entire arrays of data without having to write loops. 3) Tools for reading/writing array data to disk and working with memory mapped files. 4) Linear algebra, random number generation, and Fourier transform capabilities. 5) A C API for connecting Numpy with libraries written in C, C++ or FORTRAN.

In [1]:

```
import numpy as np
```

In [2]:

```
a = np.array([1,2,3])

print(type(a))
```

```
<class 'numpy.ndarray'>
```

In [3]:

```
print(np.shape(a))
```

```
(3,)
```

So it indicates it is one dimensional array with three values. This is how we create an array with one dimmension. Now lets go further to create multidimensional arrays. You have to nest the elements.

In [6]:

```
b = np.array([[1,2],[3,4]])
print(b)
print(np.shape(b))
```

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

So it is a two by two array with elements 1, 2, 3, 4. See it is that simple with numpy library. And don't forget we have to use the dot function to access the NumPy library functions.

In [7]:

```
b[1,0]
```

Out[7]:

```
3
```

One thing I also want to include is how do we access these values. So if I wanted to access let's say in row 2 I wanted to access the first value which would be 3 right here at the output. You would index, specify in the row first, which in our case would be 1 because Python indexes from starting with 0 for position 1. So it'd be 1, 0 and if we did this correctly it should output 3, and it does.

Now, one of the additional functionalities of NumPy in arrays is that you can create matrices that include all 0s, all 1s. And you can also create a full constant array with a specified value.

In [11]:

```
c = np.zeros((3,2))
print(c)
d = np.ones((2,2))
print(d)
```

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

Now lets print an array with random values.

In [15]:

```
data = np.random.randn(2,3)
print(data)
```

```
[[ 1.69787209  0.47733923 -0.65562252]
 [-0.99495979 -2.12571176 -0.83347407]]
```

So now I'll show you how you create a constant array with a specified value. And you do this by using the np full function, which takes 2 inputs. The first will be your dimensions so we'll do a 3 by 3 array. And then we also want to say 7 as our values for each position. And if we go ahead and run that, print it out.

In [17]:

```
e = np.full((3,3),7)
print(e)
```

```
[[7 7 7]
 [7 7 7]
 [7 7 7]]
```

Array Indexing:

In [18]:

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

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

Now, how do we go about indexing and splicing this array? We'll first start with the rows that we would like. And so, we'll do position 0 and 1 and then, our range, 2, specifies that we want to end that 2. So, we'll only take the first 2 rows. And then we'll do our columns. And we'll start at position 2, and end at 3, which means we want position 1, which is actually column 2 and column 3. And then position 4 is our endpoint so we do not access it. So if we want to go now and print out our new value, of h, here we are.

In [19]:

```
h = h[:2,1:3]
print(h)
```

```
[[2 3]
 [6 7]]
```

Datatypes in Array: So there are only two types since we are dealing with mathematical operations i.e. integer and float type. So lets create both of these types.

In [21]:

```
i = np.array([1,2])

print(i.dtype)

j = np.array([1.25, 0.2568])

print(j.dtype)
```

```
int64
float64
```

The dtype function is an object describing the data type of the array. It is clearly differnbt than the type object used earlier. We can also force the data type into our array. For example we can convert float datatype to integer.

In [26]:

```
k = np.array([1.25, 0.2568], dtype = np.int64)

print(k.dtype)
```

```
int64
```

Array Mathematics: Numpy can do basic as well as advanced maths with the array. It can multiply, divide, add, and substract arrays. It is eay with Numpy.

In [30]:

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

print(x + y)
print(x - y)
print(x * y)
print(x / y)
```

```
[[ 8 10]
 [12 14]
 [10 12]]
[[-6 -6]
 [-6 -6]
 [ 0  0]]
[[ 7 16]
 [27 40]
 [25 36]]
[[0.14285714 0.25      ]
 [0.33333333 0.4       ]
 [1.         1.        ]]
```

We can also do square root of the elements in an array

In [31]:

```
print(np.sqrt(x))
```

```
[[1.         1.41421356]
 [1.73205081 2.        ]
 [2.23606798 2.44948974]]
```

Additionally NumPy includes some descriptive statistics. For example, we can use the sum function or mean function and a numerous amount of others that NumPy provides, and I do recommend looking at the documentation for all of these.

In [32]:

```
np.sum(x)
```

Out[32]:

```
21
```

Now we can actually slice this by doing an axis parameter and specifying if we want to do, just sum up the rows or the individual columns.

In [33]:

```
np.sum(x,axis=0)
```

Out[33]:

```
array([ 9, 12])
```

In [34]:

```
np.sum(x,axis=1)
```

Out[34]:

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

In [ ]:

```

```