Numpy Crash Course

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

Filtering

We can also use boolean indexing/masks. Suppose we want to set all elements greater than MAX to MAX:

In [5]:
```
MAX = 5
nums = np.array([1, 4, 10, -1, 15, 0, 5])
print(nums > MAX)


nums[nums > MAX] = MAX
print(nums)
```

```
[False False  True False  True False False]
[ 1  4  5 -1  5  0  5]
```

Stacking

`numpy.stack()` : **Joins a sequence of arrays along a new axis.**

In [6]:
```
a =  np.arange(9).reshape(3 , 3)
print(a)
```

```
[[0 1 2]
 [3 4 5]
 [6 7 8]]
```

In [7]:
```
b = np.arange(10,19).reshape(3,3)
print(b)
```

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

In [8]:
```
c = np.arange(20,29).reshape(3,3)
print(c)
```

```
[[20 21 22]
 [23 24 25]
 [26 27 28]]
```

Horizontal Stacking

In [9]:
```
np.hstack((a, b,c))
```

Out[9]:
```
array([[ 0,  1,  2, 10, 11, 12, 20, 21, 22],
       [ 3,  4,  5, 13, 14, 15, 23, 24, 25],
       [ 6,  7,  8, 16, 17, 18, 26, 27, 28]])
```

Vertical Stacking

In [10]:
```
np.vstack((a, c , b))
```

Out[10]:
```
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [20, 21, 22],
       [23, 24, 25],
       [26, 27, 28],
       [10, 11, 12],
       [13, 14, 15],
       [16, 17, 18]])
```

1D vectors

The problem with 1 D vectors or 0 rank arrays is that you can't define them as row or column vectors, so sometimes it may be better to define them as 2D

In [11]:
```
v=np.random.rand(5)
print(v)
print(v.shape)
```

```
[0.99399181 0.88830143 0.21620255 0.5822636  0.78464844]
(5,)
```

Row Vector

In [12]:
```
v=np.random.rand(1,5)
print(v)
print(v.shape)  #row vector
```

```
[[0.05290824 0.61838694 0.47541656 0.25096532 0.03506057]]
(1, 5)
```

Column Vector

In [13]:
```
v=np.random.rand(5,1)
print(v)
print(v.shape)   #column vector
```

```
[[0.82158503]
 [0.38584894]
 [0.68100555]
 [0.27551084]
 [0.53602123]]
(5, 1)
```

Converting Python Lists to Numpy arrays

In [15]:
```
pythonlist = [2,3,4,4]
print(type(pythonlist))
numpylist = np.array(pythonlist)
print(type(numpylist))
```

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

View and Copies.

Unlike a copy, in a **view** of an array, the data is shared between the view and the array. Sometimes, our results are copies of arrays, but other times they can be views. Understanding when each is generated is important to avoid any unforeseen issues.

Views

Views can be created from a slice of an array

In [16]:
```
x = np.arange(5)
print('Original:\n', x)

# Modifying the view will modify the array
view = x[0:3]
view[0] = -100
print('Array After Modified View:\n', x)
```

```
Original:
 [0 1 2 3 4]
Array After Modified View:
 [-100    1    2    3    4]
```

Copy

Just add a `.copy()` after the array to prevent it from being modified

In [17]:
```
x = np.arange(5)
print('Original:\n', x)

# Copy, will not modify the original array.
copy = x[1:3].copy()
copy[1] = -100
print('Copy:\n', copy)
print('Array After Modified Copy:\n', x)
```

```
Original:
 [0 1 2 3 4]
Copy:
 [   1 -100]
Array After Modified Copy:
 [0 1 2 3 4]
```

Summary

1. Numpy is an incredibly powerful library for computation providing both massive efficiency gains and convenience.
2. Vectorize! Orders of magnitude faster.
3. Keeping track of the shape of your arrays is often useful.
4. Many of the useful math functions and operations are built into Numpy.
5. Watch out for views vs. copies.