NumPy Indexing and Selection

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

In [3]:
```
#Creating sample array
arr = np.arange(0,11)
```

In [4]:
```
#Show
arr
```

Out[4]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

Bracket Indexing and Selection

The simplest way to pick one or some elements of an array looks very similar to python lists:

In [5]:
```
#Get a value at an index
arr[8]
```

Out[5]:
```
8
```

In [6]:
```
#Get values in a range
arr[1:5]
```

Out[6]:
```
array([1, 2, 3, 4])
```

In [7]:
```
#Get values in a range
arr[0:5]
```

Out[7]:
```
array([0, 1, 2, 3, 4])
```

Broadcasting

Numpy arrays differ from a normal Python list because of their ability to broadcast:

In [8]:
```
#Setting a value with index range (Broadcasting)
arr[0:5]=100

#Show
arr
```

Out[8]:
```
array([100, 100, 100, 100, 100,   5,   6,   7,   8,   9,  10])
```

In [9]:
```
# Reset array, we'll see why I had to reset in  a moment
arr = np.arange(0,11)

#Show
arr
```

Out[9]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

In [10]:
```
#Important notes on Slices
slice_of_arr = arr[0:6]

#Show slice
slice_of_arr
```

Out[10]:
```
array([0, 1, 2, 3, 4, 5])
```

In [11]:
```
#Change Slice
slice_of_arr[:]=99

#Show Slice again
slice_of_arr
```

Out[11]:
```
array([99, 99, 99, 99, 99, 99])
```

Now note the changes also occur in our original array!

In [12]:
```
arr
```

Out[12]:
```
array([99, 99, 99, 99, 99, 99,  6,  7,  8,  9, 10])
```

Data is not copied, it's a view of the original array! This avoids memory problems!

In [13]:
```
#To get a copy, need to be explicit
arr_copy = arr.copy()

arr_copy
```

Out[13]:
```
array([99, 99, 99, 99, 99, 99,  6,  7,  8,  9, 10])
```

Indexing a 2D array (matrices)

The general format is **arr_2d[row][col]** or **arr_2d[row,col]**. I recommend usually using the comma notation for clarity.

In [11]:
```
arr_2d = np.array(([5,10,15],[20,25,30],[35,40,45]))

#Show
arr_2d
```

Out[11]:
```
array([[ 5, 10, 15],
       [20, 25, 30],
       [35, 40, 45]])
```

In [8]:
```
#Indexing row
arr_2d[1]
```

Out[8]:
```
array([20, 25, 30])
```

In [9]:
```
# Format is arr_2d[row][col] or arr_2d[row,col]

# Getting individual element value
arr_2d[1][0]
```

Out[9]:
```
20
```

In [10]:
```
# Getting individual element value
arr_2d[1,0]
```

Out[10]:
```
20
```

In [18]:
```
# 2D array slicing

#Shape (2,2) from top right corner
arr_2d[:2,1:]
```

Out[18]:
```
array([[10, 15],
       [25, 30]])
```

In [19]:
```
#Shape bottom row
arr_2d[2]
```

Out[19]:
```
array([35, 40, 45])
```

In [20]:
```
#Shape bottom row
arr_2d[2,:]
```

Out[20]:
```
array([35, 40, 45])
```

Selection

Let's briefly go over how to use brackets for selection based off of comparison operators.

In [28]:
```
arr = np.arange(1,11)
arr
```

Out[28]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

In [30]:
```
arr > 4
```

Out[30]:
```
array([False, False, False, False,  True,  True,  True,  True,  True,  True], d
type=bool)
```

In [31]:
```
bool_arr = arr>4
```

In [32]:
```
bool_arr
```

Out[32]:
```
array([False, False, False, False,  True,  True,  True,  True,  True,  True], d
type=bool)
```

In [33]:
```
arr[bool_arr]
```

Out[33]:
```
array([ 5,  6,  7,  8,  9, 10])
```

In [34]:
```
arr[arr>2]
```

Out[34]:
```
array([ 3,  4,  5,  6,  7,  8,  9, 10])
```

In [37]:
```
x = 2
arr[arr>x]
```

Out[37]:
```
array([ 3,  4,  5,  6,  7,  8,  9, 10])
```