**NumPy Indexing and Selection**[**¶**](#gjdgxs)

In this lecture we will discuss how to select elements or groups of elements from an array.

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**[**¶**](#30j0zll)

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**[**¶**](#1fob9te)

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)**[**¶**](#3znysh7)

The general format is **arr\_2d[row][col]** or **arr\_2d[row,col]**. I recommend usually using the comma notation for clarity.

In [14]:

arr\_2d = np.array(([5,10,15],[20,25,30],[35,40,45]))  
  
*#Show*  
arr\_2d

Out[14]:

array([[ 5, 10, 15],  
 [20, 25, 30],  
 [35, 40, 45]])

In [15]:

*#Indexing row*  
arr\_2d[1]

Out[15]:

array([20, 25, 30])

In [16]:

*# Format is arr\_2d[row][col] or arr\_2d[row,col]*  
  
*# Getting individual element value*  
arr\_2d[1][0]

Out[16]:

20

In [17]:

*# Getting individual element value*  
arr\_2d[1,0]

Out[17]:

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])

**Fancy Indexing**[**¶**](#2et92p0)

Fancy indexing allows you to select entire rows or columns out of order,to show this, let's quickly build out a numpy array:

In [21]:

*#Set up matrix*  
arr2d = np.zeros((10,10))

In [22]:

*#Length of array*  
arr\_length = arr2d.shape[1]

In [23]:

*#Set up array*  
  
**for** i **in** range(arr\_length):  
 arr2d[i] = i  
   
arr2d

Out[23]:

array([[ 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],  
 [ 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],  
 [ 2., 2., 2., 2., 2., 2., 2., 2., 2., 2.],  
 [ 3., 3., 3., 3., 3., 3., 3., 3., 3., 3.],  
 [ 4., 4., 4., 4., 4., 4., 4., 4., 4., 4.],  
 [ 5., 5., 5., 5., 5., 5., 5., 5., 5., 5.],  
 [ 6., 6., 6., 6., 6., 6., 6., 6., 6., 6.],  
 [ 7., 7., 7., 7., 7., 7., 7., 7., 7., 7.],  
 [ 8., 8., 8., 8., 8., 8., 8., 8., 8., 8.],  
 [ 9., 9., 9., 9., 9., 9., 9., 9., 9., 9.]])

Fancy indexing allows the following

In [24]:

arr2d[[2,4,6,8]]

Out[24]:

array([[ 2., 2., 2., 2., 2., 2., 2., 2., 2., 2.],  
 [ 4., 4., 4., 4., 4., 4., 4., 4., 4., 4.],  
 [ 6., 6., 6., 6., 6., 6., 6., 6., 6., 6.],  
 [ 8., 8., 8., 8., 8., 8., 8., 8., 8., 8.]])

In [25]:

*#Allows in any order*  
arr2d[[6,4,2,7]]

Out[25]:

array([[ 6., 6., 6., 6., 6., 6., 6., 6., 6., 6.],  
 [ 4., 4., 4., 4., 4., 4., 4., 4., 4., 4.],  
 [ 2., 2., 2., 2., 2., 2., 2., 2., 2., 2.],  
 [ 7., 7., 7., 7., 7., 7., 7., 7., 7., 7.]])

**More Indexing Help**[**¶**](#tyjcwt)

Indexing a 2d matrix can be a bit confusing at first, especially when you start to add in step size. Try google image searching NumPy indexing to fins useful images, like this one:

<img src= '<http://memory.osu.edu/classes/python/_images/numpy_indexing.png>' width=500/>

**Selection**[**¶**](#3dy6vkm)

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], dtype=bool)

In [31]:

bool\_arr = arr>4

In [32]:

bool\_arr

Out[32]:

array([False, False, False, False, True, True, True, True, True, True], dtype=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])