NumPy Indexing and Selection

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

In [1]:

```
import numpy as np
```

In [6]:

```
#Creating sample array
arr = np.arange(10,20)
```

In [7]:

```
#Show
arr
```

Out[7]:

```
array([10, 11, 12, 13, 14, 15, 16, 17, 18, 19])
```

Bracket Indexing and Selection

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

In [8]:

```
arr[2]
```

Out[8]:

```
12
```

In [9]:

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

Out[9]:

```
18
```

In [10]:

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

Out[10]:

```
array([11, 12, 13, 14])
```

In [12]:

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

Out[12]:

```
array([10, 11, 12, 13, 14])
```

In [13]:

```
arr[0:5] = 500
```

In [14]:

```
arr
```

Out[14]:

```
array([500, 500, 500, 500, 500,  15,  16,  17,  18,  19])
```

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 [16]:

```
#Important notes on Slices
slice_of_arr = arr[5:10]

#Show slice
slice_of_arr
```

Out[16]:

```
array([15, 16, 17, 18, 19])
```

In [17]:

```
slice_of_arr[:] = 100
```

In [19]:

```
slice_of_arr
```

Out[19]:

```
array([100, 100, 100, 100, 100])
```

In [20]:

```
arr
```

Out[20]:

```
array([500, 500, 500, 500, 500, 100, 100, 100, 100, 100])
```

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 [21]:

```
arr
```

Out[21]:

```
array([500, 500, 500, 500, 500, 100, 100, 100, 100, 100])
```

In [22]:

```
arrcopy = arr.copy()
```

In [23]:

```
arr
```

Out[23]:

```
array([500, 500, 500, 500, 500, 100, 100, 100, 100, 100])
```

In [24]:

```
arrcopy
```

Out[24]:

```
array([500, 500, 500, 500, 500, 100, 100, 100, 100, 100])
```

In [25]:

```
arrcopy[0:3] = 1000
```

In [26]:

```
arrcopy
```

Out[26]:

```
array([1000, 1000, 1000,  500,  500,  100,  100,  100,  100,  100])
```

In [27]:

```
arr
```

Out[27]:

```
array([500, 500, 500, 500, 500, 100, 100, 100, 100, 100])
```

In [ ]:

```

```

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

In [28]:

```
#To get a copy, need to be explicit
arr_copy = arr.copy()

arr_copy
```

Out[28]:

```
array([500, 500, 500, 500, 500, 100, 100, 100, 100, 100])
```

In [ ]:

```

```

In [30]:

```
new_arr = np.array(([10,20,30],[40,50,60],[70,80,90]))

print(new_arr)
```

```
[[10 20 30]
 [40 50 60]
 [70 80 90]]
```

In [35]:

```
new_arr[0]
```

Out[35]:

```
array([10, 20, 30])
```

In [32]:

```
1 new_arr[1]
```

Out[32]:

```
array([40, 50, 60])
```

In [33]:

```
1 new_arr[2]
```

Out[33]:

```
array([70, 80, 90])
```

In [ ]:

```
1
```

In [36]:

```
1 new_arr[0][1]
```

Out[36]:

```
20
```

In [37]:

```
1 new_arr[1][2]
```

Out[37]:

```
60
```

In [38]:

```
1 new_arr[2][2]
```

Out[38]:

```
90
```

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 [39]:

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

#Show
arr_2d
```

Out[39]:

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

In [40]:

```
#Indexing row
arr_2d[1]

```

Out[40]:

```
array([20, 25, 30])
```

In [41]:

```
# Format is arr_2d[row][col] or arr_2d[row,col]

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

Out[41]:

```
20
```

In [43]:

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

Out[43]:

```
20
```

In [57]:

```
arr_2d
```

Out[57]:

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

In [63]:

```
arr_2d[:2,0:2]
```

Out[63]:

```
array([[ 5, 10],
       [20, 25]])
```

In [65]:

```
arr_2d[0:2,0:2]
```

Out[65]:

```
array([[ 5, 10],
       [20, 25]])
```

In [62]:

```
arr_2d[0:2,0:2]
```

Out[62]:

```
array([[ 5, 10],
       [20, 25]])
```

In [ ]:

```

```

In [52]:

```
arr_2d[:2,1:]
```

Out[52]:

```
array([[10, 15],
       [25, 30]])
```

In [53]:

```
arr_2d[:2][:2]
```

Out[53]:

```
array([[ 5, 10, 15],
       [20, 25, 30]])
```

In [55]:

```
arr_2d[:1][:1]
```

Out[55]:

```
array([[ 5, 10, 15]])
```

In [56]:

```
arr_2d[:1][0:1]
```

Out[56]:

```
array([[ 5, 10, 15]])
```

In [ ]:

```
arr_2d[]
```

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

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 [ ]:

```

```

In [67]:

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

In [69]:

```
arr2d
```

Out[69]:

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

In [71]:

```
arr2d.shape
```

Out[71]:

```
(10, 10)
```

In [72]:

```
#Length of array
arr_length = arr2d.shape[1]
```

In [75]:

```
arr_length
```

Out[75]:

```
10
```

In [77]:

```
for i in range(0,arr_length):
    arr2d[i] = i
print(arr2d)
```

```
[[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.]]
```

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.]])
```

In [79]:

```
arr2d[[1,3,5,7,9]]
```

Out[79]:

```
array([[1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],
       [3., 3., 3., 3., 3., 3., 3., 3., 3., 3.],
       [5., 5., 5., 5., 5., 5., 5., 5., 5., 5.],
       [7., 7., 7., 7., 7., 7., 7., 7., 7., 7.],
       [9., 9., 9., 9., 9., 9., 9., 9., 9., 9.]])
```

In [80]:

```
arr2d[3]
```

Out[80]:

```
array([3., 3., 3., 3., 3., 3., 3., 3., 3., 3.])
```

Fancy indexing allows the following

In [81]:

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

Out[81]:

```
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 [ ]:

```

```

In [82]:

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

Out[82]:

```
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.]])
```

In [ ]:

```

```

More Indexing Help

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:

Selection

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

In [83]:

```
arr = np.arange(1,11)
arr
```

Out[83]:

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

In [84]:

```
arr > 5
```

Out[84]:

```
array([False, False, False, False, False,  True,  True,  True,  True,
        True])
```

In [85]:

```
bool_arr = arr>5
```

In [86]:

```
bool_arr
```

Out[86]:

```
array([False, False, False, False, False,  True,  True,  True,  True,
        True])
```

In [88]:

```
arr[bool_arr]
```

Out[88]:

```
array([ 6,  7,  8,  9, 10])
```

In [90]:

```
arr[arr>5]
```

Out[90]:

```
array([ 6,  7,  8,  9, 10])
```

In [91]:

```
val =6

arr[arr>val]
```

Out[91]:

```
array([ 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])
```

Type *Markdown* and LaTeX: α^2

In [ ]:

```

```