# NumPyTraining

October 7, 2020

[1]:
```
import numpy as np
```

## 0.1 Basics of NumPy

Creation of Single Dimensional Array

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

```
[1 2 3]
```

Creation of Two Dimensional Array

[3]:
```
b = np.array([[1,2,3],[4,5,6],[7.0,8.0,9.0]])
print(b)
```

```
[[1. 2. 3.]
 [4. 5. 6.]
 [7. 8. 9.]]
```

Creation of Three Dimensional Array

[4]:
```
c = np.
 ↪array([[[1,2,3],[4,5,6],[7,8,9]],[[1,2,3],[4,5,6],[7,8,9]],[[1,2,3],[4,5,6],[7,8,9]]])
print(c)
```

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

 [[1 2 3]
  [4 5 6]
  [7 8 9]]

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

Getting a Dimensiona of an Array

[5]:
```
print(a.ndim)
print(b.ndim)
print(c.ndim)
```

```
1
2
3
```

Getting a Shape of an Array

[6]:
```
print(a.shape)
print(b.shape)
print(c.shape)
```

```
(3,)
(3, 3)
(3, 3, 3)
```

Getting a Type of an Array

[7]:
```
print(a.dtype)
print(b.dtype)
```

```
int64
float64
```

[8]:
```
a = np.array([1,2,3], dtype = 'int16')
print(a)
```

```
[1 2 3]
```

[9]:
```
print(a.dtype)
```

```
int16
```

Getting Size of an Array

[10]:
```
print(a.itemsize)
print(b.itemsize)
print(c.itemsize)
```

```
2
8
8
```

Getting Total No. of elements in an Array

[11]:
```
print(a.size)
print(b.size)
print(c.size)
```

```
3
9
27
```

Getting Total Size of an Array

[12]:
```
print(a.nbytes)
print(b.nbytes)
```

```
6
72
```

### 0.1.1 Accessing/ Changing Specific Elements, Rows, Columns etc

*1-d and 2-d example

[13]:
```
a = np.array([[1,2,3,4,5,6,7],[11,12,13,14,15,16,17]])
print(a)
print(a.ndim)
```

```
[[ 1  2  3  4  5  6  7]
 [11 12 13 14 15 16 17]]
2
```

Getting a Specific Element [r, c]

[14]:
```
a[1,5]
```

[14]:
```
16
```

[15]:
```
a[1,-2]
```

[15]:
```
16
```

Getting a Specific Row

[16]:
```
a[0,:]
```

[16]:
```
array([1, 2, 3, 4, 5, 6, 7])
```

Getting a Specific Column

[17]:
```
a[:, 3]
```

[17]:
```
array([ 4, 14])
```

Getting a Specific Elements in an Array

[18]:
```
a[0, 1:4]
```

[18]:
```
array([2, 3, 4])
```

[19]:
```
# We can Stepsize also
a[0, 1:6:2]
```

[19]:
```
array([2, 4, 6])
```

[20]:
```
a[:, 1:6]
```

[20]:
```
array([[ 2,  3,  4,  5,  6],
       [12, 13, 14, 15, 16]])
```

Changing Element Values in an Array

[21]:
```
print(a)
```

```
[[ 1  2  3  4  5  6  7]
 [11 12 13 14 15 16 17]]
```

[22]:
```
a[1,5] = 20
```

[23]:
```
print(a)
```

```
[[ 1  2  3  4  5  6  7]
 [11 12 13 14 15 20 17]]
```

Changing Values in a Particular Column

[24]:
```
print(a[:,2])
```

```
[ 3 13]
```

[25]:
```
a[:,2] = 5
print(a)
```

```
[[ 1  2  5  4  5  6  7]
 [11 12  5 14 15 20 17]]
```

[26]:
```
a[:,2] = [1,9]
print(a)
```

```
[[ 1  2  1  4  5  6  7]
 [11 12  9 14 15 20 17]]
```

Changing Values in Particular Row

[27]:
```
print(a)
print(a[0,:])
```

```
[[ 1  2  1  4  5  6  7]
 [11 12  9 14 15 20 17]]
[1 2 1 4 5 6 7]
```

[28]:
```
a[0,:] = [21,22,23,24,25,26,27]
print(a)
```

```
[[21 22 23 24 25 26 27]
 [11 12  9 14 15 20 17]]
```

#3-d example

[29]:
```
c = np.
 ↪array([[[1,2,3],[4,5,6],[7,8,9]],[[11,12,13],[14,15,16],[17,18,19]],[[21,22,23],[24,25,26],
print(c)
```

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

 [[11 12 13]
  [14 15 16]
  [17 18 19]]

 [[21 22 23]
  [24 25 26]
  [27 28 29]]]
```

Getting a Specific Element from a 3-d Array [a, r, c]

[30]:
```
c[0,1,1]
```

[30]:
```
5
```

[31]:
```
c[:,1,:]
```

[31]:
```
array([[ 4,  5,  6],
       [14, 15, 16],
       [24, 25, 26]])
```

Changing or Replacing elements in 3-d array

[32]:
```
print(c)
```

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

 [[11 12 13]
  [14 15 16]
```

```
 [17 18 19]]

[[21 22 23]
 [24 25 26]
 [27 28 29]]]
```

[33]:
```
c[0,1,1] = 55
print(c)
```

```
[[[ 1  2  3]
  [ 4 55  6]
  [ 7  8  9]]

 [[11 12 13]
  [14 15 16]
  [17 18 19]]

 [[21 22 23]
  [24 25 26]
  [27 28 29]]]
```

[34]:
```
c[:,1,:]
```

[34]:
```
array([[ 4, 55,  6],
       [14, 15, 16],
       [24, 25, 26]])
```

[35]:
```
c[:,1,:] = [[51,52,53],[61,62,63],[71,72,73]]
```

[36]:
```
c
```

[36]:
```
array([[[ 1,  2,  3],
        [51, 52, 53],
        [ 7,  8,  9]],

       [[11, 12, 13],
        [61, 62, 63],
        [17, 18, 19]],

       [[21, 22, 23],
        [71, 72, 73],
        [27, 28, 29]]])
```

### 0.1.2 Initializing Different Types of Arrays

Zeroes 0's Matrix

[37]:
```
z = np.zeros((3,3))
print(z)
```

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

One 1's Matrix

[38]:
```
o = np.ones((3,3,3))
print(o)
```

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

 [[1. 1. 1.]
  [1. 1. 1.]
  [1. 1. 1.]]

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

Other Number Matrix

[39]:
```
n = np.full((3,3),99, dtype='float32')
print(n)
```

```
[[99. 99. 99.]
 [99. 99. 99.]
 [99. 99. 99.]]
```

Mimicing a Array and fill with Other Number (Full_like)

[40]:
```
a
```

[40]:
```
array([[21, 22, 23, 24, 25, 26, 27],
       [11, 12,  9, 14, 15, 20, 17]])
```

[41]:
```
k = np.full_like(a, 5)
print(k)
```

```
[[5 5 5 5 5 5 5]
 [5 5 5 5 5 5 5]]
```

[42]:
```
l = np.full(a.shape, 9)
print(l)
```

```
[[9 9 9 9 9 9 9]
 [9 9 9 9 9 9 9]]
```

Matrix of Random Numbers

[43]:
```
i = np.random.rand(4,2)
print(i)
```

```
[[0.39350493 0.50342966]
 [0.48607076 0.34773727]
 [0.22587524 0.48777289]
 [0.33817707 0.09892646]]
```

[44]:
```
# Generating a Matrix of available Matrics shape filled with Random Data
j = np.random.random_sample(a.shape)
print(j)
```

```
[[0.07283492 0.7808883  0.32951978 0.56694182 0.6057634  0.1329585
  0.86793408]
 [0.11546676 0.57821088 0.94089756 0.62421606 0.27726301 0.14598357
  0.25742301]]
```

[45]:
```
# Random Integer Values np.random.randint(startrange,stoprange,size)
np.random.randint(6,10, size=(3,3))
```

[45]:
```
array([[6, 9, 9],
       [6, 9, 6],
       [9, 6, 7]])
```

Identity Matrix

[46]:
```
np.identity(5)
```

[46]:
```
array([[1., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [0., 0., 1., 0., 0.],
       [0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 1.]])
```

Repeating a Array

[47]:
```
arr = np.array([1,2,3])
r1 = np.repeat(arr,3)
print(r1)
```

```
[1 1 1 2 2 2 3 3 3]
```

[48]:
```
arr1 = np.array([[1,2,3]])
r2 = np.repeat(arr1,3, axis=0)
```

```
print(r2)
```

```
[[1 2 3]
 [1 2 3]
 [1 2 3]]
```

### 0.1.3 Simple Exercise Task 1

[49]:
```
output = np.ones((5,5))
# print(output)

z = np.zeros((3,3))
z[1,1] = 9
# print(z)

output[1:4,1:4] =z
print(output)
```

```
[[1. 1. 1. 1. 1.]
 [1. 0. 0. 0. 1.]
 [1. 0. 9. 0. 1.]
 [1. 0. 0. 0. 1.]
 [1. 1. 1. 1. 1.]]
```

### 0.1.4 Copying Arrays

[50]:
```
a = np.array([1,2,3])
```

[51]:
```
b = a
```

[52]:
```
b
```

[52]:
```
array([1, 2, 3])
```

[53]:
```
b[0] = 100
```

[54]:
```
b
```

[54]:
```
array([100,   2,   3])
```

[55]:
```
a
```

[55]:
```
array([100,   2,   3])
```

[56]:
```
c = a.copy()
a
```

[56]:
```
array([100,   2,   3])
```

[57]:
```
c[1] = 200
```

[58]:
```
c
```

[58]:
```
array([100, 200,   3])
```

[59]:
```
a
```

[59]:
```
array([100,   2,   3])
```

**Mathematics** Elementwise Addition, Substraction, Multiplication and Division

[60]:
```
a = np.array([1,2,3,4,5])
print(a)
```

```
[1 2 3 4 5]
```

[61]:
```
a + 2
```

[61]:
```
array([3, 4, 5, 6, 7])
```

[62]:
```
a - 2
```

[62]:
```
array([-1,  0,  1,  2,  3])
```

[63]:
```
a * 2
```

[63]:
```
array([ 2,  4,  6,  8, 10])
```

[64]:
```
a / 2
```

[64]:
```
array([0.5, 1. , 1.5, 2. , 2.5])
```

[65]:
```
a ** 2
```

[65]:
```
array([ 1,  4,  9, 16, 25])
```

[66]:
```
b = np.array([1,2,3,4,5])
```

[67]:
```
a + b
```

[67]:
```
array([ 2,  4,  6,  8, 10])
```

[68]:
```
print(np.cos(a))
print(np.sin(a))
```

```
[ 0.54030231 -0.41614684 -0.9899925  -0.65364362  0.28366219]
[ 0.84147098  0.90929743  0.14112001 -0.7568025  -0.95892427]
```

**Linear Algebra**

[69]:
```
a = np.ones((2,3))
print(a)
```

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

[70]:
```
b = np.full((3,2), 2)
print(b)
```

```
[[2 2]
 [2 2]
 [2 2]]
```

[71]:
```
np.matmul(a,b)
```

[71]:
```
array([[6., 6.],
       [6., 6.]])
```

Finding a Determinant of Matrix

[72]:
```
c = np.identity(3)
print(c)
```

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

[73]:
```
np.linalg.det(c)
```

[73]:
```
1.0
```

Finding Dot Matrix of Two Vectors

https://numpy.org/doc/stable/reference/ For Linear Algebra Refernce

[74]:
```
x = np.array([2,7,1])
y = np.array([8,2,8])
```

[75]:
```
np.dot(x,y)
```

[75]:
```
38
```

Getting a Eigen Values of a Matrix

[76]:
```
x = np.ones((3,3))
print(x)
```

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

[77]:
```
np.linalg.eig(x)
```

[77]:
```
(array([-2.22044605e-16,  3.00000000e+00,  0.00000000e+00]),
 array([[-0.81649658,  0.57735027,  0.        ],
        [ 0.40824829,  0.57735027, -0.70710678],
        [ 0.40824829,  0.57735027,  0.70710678]]))
```

**Statistics**

[78]:
```
b = np.array([[1,2,3],[4,5,6],[7.0,8.0,9.0]])
print(b)
```

```
[[1. 2. 3.]
 [4. 5. 6.]
 [7. 8. 9.]]
```

[79]:
```
np.min(b)
```

[79]: 1.0

[80]:
```
np.max(b)
```

[80]: 9.0

[81]:
```
c = np.array([1,2,3,4])
```

[82]:
```
np.mean(c)
```

[82]: 2.5

[83]:
```
np.median(c)
```

[83]: 2.5

[84]:
```
np.sum(c)
```

[84]: 10

ReOrganizing Arrays

[85]:
```
a = np.array([[1,2,3],[4,5,6]])
print(a)
```

```
[[1 2 3]
 [4 5 6]]
```

[86]:
```
g = a.reshape((3,2))
```

[87]:
```
g
```

[87]:
```
array([[1, 2],
       [3, 4],
       [5, 6]])
```

Vertically Stacking

[88]:
```
v1 = np.array([1,2,3,4])
v2 = np.array([5,6,7,8])

np.stack([v1, v2])
```

[88]:
```
array([[1, 2, 3, 4],
       [5, 6, 7, 8]])
```

Horizontal Stacking

[89]:
```
h1 = np.ones((2,4))
h2 = np.zeros((2,4))

np.hstack((h1, h2))
```

[89]:
```
array([[1., 1., 1., 1., 0., 0., 0., 0.],
       [1., 1., 1., 1., 0., 0., 0., 0.]])
```

**Concatination**

[90]:
```
n1_1dim = np.array([[1,2,3,4],[5,6,7,8]])
n2_1dim = np.array([[4,5,6,7],[7,8,9,5]])
np.concatenate([n1_1dim,n2_1dim])
```

[90]:
```
array([[1, 2, 3, 4],
       [5, 6, 7, 8],
       [4, 5, 6, 7],
       [7, 8, 9, 5]])
```

[91]:
```
np.concatenate([n1_1dim,n2_1dim], axis=0)
```

[91]:
```
array([[1, 2, 3, 4],
       [5, 6, 7, 8],
       [4, 5, 6, 7],
       [7, 8, 9, 5]])
```

[92]:
```
np.concatenate([n1_1dim,n2_1dim], axis=1)
```

[92]:
```
array([[1, 2, 3, 4, 4, 5, 6, 7],
       [5, 6, 7, 8, 7, 8, 9, 5]])
```

**Miscellaneous** Loading Data from File

[93]:
```
filedata = np.genfromtxt('data.csv', delimiter=',')
```

[94]:
```
filedata
```

[94]:
```
array([[ 1.,  2.,  3.,  4.,  5.],
       [ 6.,  7.,  8.,  9., 10.],
       [11., 12., 13., 14., 15.],
       [16., 17., 18., 19., 20.]])
```

[95]:
```
filedata.astype('int32')
```

[95]:
```
array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10],
       [11, 12, 13, 14, 15],
       [16, 17, 18, 19, 20]], dtype=int32)
```

**Boolean Masking and Advanced Indexing**

[96]:
```
filedata > 10
```

[96]:
```
array([[False, False, False, False, False],
       [False, False, False, False, False],
       [ True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True]])
```

[97]:
```
filedata[filedata > 10]
```

[97]:
```
array([11., 12., 13., 14., 15., 16., 17., 18., 19., 20.])
```

[98]:
```
a = np.array([1,2,3,4,5,6,7,8,9])
a[[1,2,8]]
```

[98]:
```
array([2, 3, 9])
```

[99]:
```
np.any(filedata > 15, axis = 0)
```

[99]:
```
array([ True,  True,  True,  True,  True])
```

[100]:
```
((filedata > 5) & (filedata < 10))
```

[100]:
```
array([[False, False, False, False, False],
       [ True,  True,  True,  True, False],
       [False, False, False, False, False],
       [False, False, False, False, False]])
```

[101]:
```
(~((filedata > 5) & (filedata < 10)))
```

[101]:
```
array([[ True,  True,  True,  True,  True],
       [False, False, False, False,  True],
       [ True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True]])
```

[ ]: