ML LAB 01 Q1 (CS22104)

Creating ndarrays -Example 1

```
# create array using array function
data1 = [6, 7.5, 8, 0, 1]
arr1 = np.array(data1)
arr1
```

```
array([6. , 7.5, 8. , 0. , 1. ])
```

```
# Nested sequences, like a list of equal-length lists, will be converted into a multidimensional array
data2 = [[1, 2, 3, 4], [5, 6, 7, 8]]
arr2 = np.array(data2)
print(arr2)
arr2.ndim
arr2.shape
```

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

```
(2, 4)
```

```
#Unless explicitly specified, np.array tries to infer a good data type for the array that it creates
# The data type is stored in a special dtype object
print( arr1.dtype)
print( arr2.dtype)
```

```
float64
int32
```

```
arr3 = np.array([1, 2, 3], dtype=np.float64)
print(arr3)
print( arr3.dtype)
```

```
[1. 2. 3.]
float64
```

```
arr4 = np.array([1, 2, 3], dtype=np.int32)
print(arr4)
print(arr4.dtype)
```

```
[1 2 3]
int32
```

To create a higher dimensional array with these methods, pass a tuple for the shape

```
np.zeros(10)
```

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

```
np.zeros((3, 6))
```

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

```
np.empty((2, 3, 2))  # initial garbage values are displayed
```

```
array([[[1.04415504e-311, 2.47032823e-322],
        [0.00000000e+000, 0.00000000e+000],
        [0.00000000e+000, 1.33664410e+160]],

       [[6.81378381e-091, 9.60074833e-071],
        [2.36683740e+179, 2.58279049e-057],
        [3.99910963e+252, 7.86644309e-067]]])
```

```
arr5= np.arange(15)
arr5
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14])
```

Type Conversion for ndarrays – Example 2

```
arr = np.array([1, 2, 3,4,5])
print( arr.dtype)
float_arr= arr.astype(np.float64)
print(float_arr.dtype)
```

```
int32
float64
```

Operation between Arrays and Scalars – Example 3

```
arr = np.array([[1.,2.,3.],[4.,5.,6.]])
print(arr)
print(arr*arr)
print(arr-arr)
print(1/arr)
print(arr**0.5)
```

```
[[1. 2. 3.]
 [4. 5. 6.]]
[[ 1.  4.  9.]
 [16. 25. 36.]]
[[0. 0. 0.]
 [0. 0. 0.]]
[[1.         0.5        0.33333333]
 [0.25       0.2        0.16666667]]
[[1.         1.41421356 1.73205081]
 [2.         2.23606798 2.44948974]]
```

Basic Indexing and slicing – Example 4

```
arr = np.arange(10)
print(arr)
print(arr[5])
print(arr[5:8])
arr[5:8]=12
print(arr)
```

```
[0 1 2 3 4 5 6 7 8 9]
5
[5 6 7]
[ 0  1  2  3  4 12 12 12  8  9]
```

Indexing with slices – Example 5

```
arr = np.arange(10)
arr
```

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

```
arr = np.arange(10)
arr[1:6]
```

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

Broadcasting – Example 6

```
arr = np.arange(5)
print(arr)
print(arr*4)
arr= np.random.randn(4,3)
print(arr.mean(0))
demeaned = arr - arr.mean (0)
print(demeaned)
print(demeaned.mean(0))
```

```
[0 1 2 3 4]
[ 0  4  8 12 16]
[ 1.0000129  -0.55879995  0.54374753]
[[ 0.60474265  0.46774408 -0.28392059]
 [-0.31002984  0.40642882  0.28750923]
 [ 0.01211057  0.03205832 -1.55637805]
 [-0.30682338 -0.90623122  1.55278942]]
[ 2.77555756e-17 -2.77555756e-17  5.55111512e-17]
```

Setting array value by broadcasting – Example 7

```
arr = np.zeros([4,3])
arr[:]=5
print (arr)
arr[:2]=[[-1.37],[0.509]]
print(arr)
```

```
[[5. 5. 5.]
 [5. 5. 5.]
 [5. 5. 5.]
 [5. 5. 5.]]
[[-1.37  -1.37  -1.37 ]
 [ 0.509  0.509  0.509]
 [ 5.     5.     5.   ]
 [ 5.     5.     5.   ]]
```

Boolean Indexing – Example 8

```
names = np.array(['bob','joe','will','bob','joe','will'])
data = np.random.randn(7,4)
print(names)
data
```

```
['bob' 'joe' 'will' 'bob' 'joe' 'will']
```

```
array([[ 0.59799018,  0.80944679,  0.22977165, -0.45459817],
       [-1.16347861, -1.98663618,  0.37322498, -0.97667939],
       [-0.27236373, -1.88586081,  0.62785704,  0.50492923],
       [ 0.25612646, -0.19498358, -0.39382792, -0.45428865],
       [-0.38530567,  0.10873488, -1.32140897, -1.21172239],
       [-0.65394086, -0.03912948, -1.32595506,  0.09305989],
       [ 0.11673491, -0.25529657, -1.10444604, -0.76008545]])
```

Fancy Indexing – Example 9

```
arr = np.empty((8,4))
for i in range(8):
      arr[i] =i
print(arr)
```

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

Transposing arrays and swapping axes- Example 10

```
arr = np.arange(15).reshape((3,5))
print(arr)
print(arr.T)
```

```
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]]
[[ 0  5 10]
 [ 1  6 11]
 [ 2  7 12]
 [ 3  8 13]
 [ 4  9 14]]
```

Reshaping arrays – Example 11

```
arr = np.arange(8)
print(arr)
arr.reshape((4,2))
```

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

```
array([[0, 1],
       [2, 3],
       [4, 5],
       [6, 7]])
```

Concatenating and splitting arrays – Example 12

```
arr1 = np.array([[1, 2, 3], [4,5, 6]])
arr2 = np.array([[7, 8, 9], [10,11, 12]])
print(np.concatenate([arr1,arr2],axis=0))
print(np.concatenate([arr1,arr2],axis=1))
```

```
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]
[[ 1  2  3  7  8  9]
 [ 4  5  6 10 11 12]]
```

Universal functions – Example 13

```
arr = np.arange(10)
print(np.sqrt(arr))
```

```
[0.         1.         1.41421356 1.73205081 2.         2.23606798
 2.44948974 2.64575131 2.82842712 3.        ]
```

```
arr = np.arange(10)
print(np.add.reduce(arr))
print(arr.sum())
```

```
45
45
```

Mathematical and Statistical methods – Example 14

```
arr= np.random.randn(5,4)
print (arr.mean())
print(np.mean(arr))
print(arr.sum())
```

```
0.43026398557133055
0.43026398557133055
8.605279711426611
```

Sorting – Example 15

```
arr= np.random.randn(5,4)
print(arr)
print (arr.sort())
print(np.sort(1))
```

```
[[-0.86732631  1.3898754  -0.4088187  -0.48398129]
 [ 1.41285358 -0.38417636 -1.07135747  0.24591978]
 [ 0.03512021 -0.6626756   2.61013482  0.1916116 ]
 [ 0.32100388  1.75772509 -0.11644529 -0.19204219]
 [-0.44188236 -2.23529315 -1.32361018  1.11661735]]
None
```

Unique and other set logic – Example 16

```
names = np.array (['bob','joe','will','bob','joe','will'])
print(np.unique(names))
ints= np.array([3,3,3,2,2,1,1,4,4])
print(np.unique(ints))
```

```
['bob' 'joe' 'will']
[1 2 3 4]
```

Linear algebra – Example 17

```
x= np.array([[1.,2.,3.],[4.,5.,6.]])
y = np.array ([[6.,23.],[-1,7],[8,9]])
print(x)
print(y)
print(x.dot(y))
```

```
[[1. 2. 3.]
 [4. 5. 6.]]
[[ 6. 23.]
 [-1.  7.]
 [ 8.  9.]]
[[ 28.  64.]
 [ 67. 181.]]
```

Random no generation – Example 18

```
samples = np.random.normal(size=(4,4))
samples
```

```
array([[-1.52173305, -1.54320823, -0.46345561,  0.25248226],
       [ 0.71252066,  0.38073195, -0.75839452, -1.24293497],
       [ 1.36270195, -0.48143912,  0.49366022, -1.86301328],
       [-0.2017939 ,  1.51548415, -1.41511044,  0.82049295]])
```

File input and output with arrays – Example 19

```
arr = np.arange(10)
print(np.save('some_Array',arr))
print(np.load('some_Array.npy'))
```

```
None
[0 1 2 3 4 5 6 7 8 9]
```

Saving and loading text files – Example 20

```
0.580052,0.186730,1.040717,1.134411
0.194163,-0.636917,-0.938659,0.124094
-0.126410,0.268607,-0.695724,0.047428
-1.484413,0.004176,-0.744203,0.005487
2.302869,0.200131,1.670238,-1.881090
-0.193230,1.047233,0.482803,0.960334
```

```
arr = np.loadtxt('array_ex.txt', delimiter=',')
```

```
array([[ 0.5801,  0.1867,  1.0407,  1.1344],[ 0.1942, -0.6369, -0.9387,
0.1241],[-0.1264,  0.2686, -0.6957,  0.0474],[-1.4844,  0.0042, -
0.7442,  0.0055],[ 2.3029,  0.2001,  1.6702, -1.8811],[-0.1932,  1.0472,
0.4828,  0.9603]])
```