CS 38003 PYTHON PROGRAMMING

Ruby Tahboub

NUMPY

NUMPY

- NumPy, stands for Numerical Python.
- NumPy has in-built functions for linear algebra and random number generation.
- NumPy is often used along with packages like SciPy (Scientific Python) and Matplotlib (plotting library) as replacement to Matlab.
- Why NumPy?
  - Fast.
  - Supports a lot more mathematical operation than python lists.
  - One of the main Python libraries that other libraries rely heavily on it.

NUMPY ARRAYS

- The most important object defined in NumPy is an N-dimensional array type called ndarray.
- To make array operations fast, ndarrays have fixed size and contain elements of the same data type.

```
import numpy as np

oneDimArray = np.array([1,2,3])
print("printing oneDimArray")
print (oneDimArray)

twoDimArray = np.array([[1, 2], [3, 4]])
print("printing twoDimArray")
print (twoDimArray)
```

```
printing oneDimArray
[1 2 3]

printing twoDimArray
[[1 2]
 [3 4]]
```

NUMPY ARRAY INITIALIZATION

- General form: `numpy.array(object, dtype = None)`
- **object**: any object exposing the array interface method returns an array, or any (nested) sequence.
- **dtype** (Optional): specifies data type of array, e.g., `np.dtype(np.int32)`

```
import numpy as np

myArray = np.array([1, 2, 3], dtype = complex)
print(myArray)
```

```
[1.+0.j 2.+0.j 3.+0.j]
```

NUMPY ARRAY INITIALIZATION

- Initializing an empty array

```
myArray = np.empty([3,2], dtype = int)
print (myArray)
```

```
[[          0          0]
 [ 4451926019 4451969400]
 [ 4451969472 844424930131968]]
```

- Initializing an array with zeros (default type is float)

```
zerosArray = np.zeros(5)
print(zerosArray)
```

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

```
zeros2DArray = np.zeros([5,2])
print(zeros2DArray)
```

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

- Initializing an array with ones

```
ones = np.ones(5)
print(ones)
```

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

numpy.arange

- `numpy.arange(start, stop, step, dtype)`
  - Returns an ndarry object containing evenly spaced values within a range.

```
import numpy as np

# start and stop parameters are set
myArray = np.arange(10,20,2, dtype = int)
print (myArray)
```

```
[10 12 14 16 18]
```

NUMPY ARRAY ATTRIBUTES

- `ndarray.ndim`
  - The number of axes (dimensions) of the array. In the Python world, the number of dimensions is referred to as **rank**.
- `ndarray.shape`
  - The dimensions of the array. This is a tuple of integers indicating the size of the array in each dimension.
  - For a matrix with n rows and m columns, shape will be (n,m). The length of the shape tuple is therefore the **rank**, or number of dimensions, ndim.
- `ndarray.size`
  - The total number of elements in the array. This is equal to the product of the elements of shape.

NUMPY ARRAY ATTRIBUTES

- `ndarray.reshape`
  - Reshapes the ndarray
  - Note: It’s not the same as the transpose operation.

```
import numpy as np
myArray = np.array([[1,2,3],[4,5,6]])

print (myArray)
print (myArray.ndim)
print (myArray.shape)

reshapedArray = myArray.reshape(3,2)
print (reshapedArray)
print (reshapedArray.ndim)
print (reshapedArray.shape)
```

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

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

`reshape` EXAMPLE

```
import numpy as np

myArray = np.arange(24)
print (myArray)
print (myArray.ndim)

# reshaping myArray
reshapedArray = myArray.reshape(2,4,3)

# reshapedArray now has three dimensions
print (reshapedArray)
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23]

1

[[[ 0  1  2]
  [ 3  4  5]
  [ 6  7  8]
  [ 9 10 11]]

 [[12 13 14]
  [15 16 17]
  [18 19 20]
  [21 22 23]]]
```

NUMPY INDEXING AND SLICING

- The content of an ndarray object can be accessed and modified by indexing and slicing.

```
import numpy as np

myArray = np.arange(10)
print (myArray)

mySlice = slice(2,7,2)
print (mySlice)
print (myArray[mySlice])
print (myArray[range(2,7,2)])
```

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

slice(2, 7, 2)

[2 4 6]

[2 4 6]
```

NUMPY INDEXING AND SLICING

```
import numpy as np

myArray = np.array([[1,2,3],[3,4,5],[4,5,6]])
print (myArray)
print ('myArray[2,1]=', myArray[2,1])

# slice items starting from index
print ('Slicing myArray from the index a[1:]')
print (myArray[1:])
```

```
[[1 2 3]
 [3 4 5]
 [4 5 6]]
myArray[2,1]= 5
Slicing myArray from the index a[1:]
[[3 4 5]
 [4 5 6]]
```

ELLIPSIS

- Ellipsis is used for slicing multidimensional numpy arrays.

```
import numpy as np

myArray = np.array([[1,2,3],[3,4,5],[4,5,6]])
print ('printing myArray')
print (myArray)

print ('second row:', myArray[1])
print ('all rows up to 2 (exclusive)', myArray[:2])

# this returns array of items in the second column
print ('The items in the second column are:')
print (myArray[...,1]) # equivalent to myArray[:,1]

# Now we will slice all items from the second row
print ('The items in the second row are:')
print (myArray[1,...])

print (myArray[1:])

# Now we will slice all items from column 1 onwards
print ('The items column 1 onwards are:')
print (myArray[...,1:])
```

```
[[1 2 3]
 [3 4 5]
 [4 5 6]]
second row: [3 4 5]
all rows up to 2 (exclusive)
 [[1 2 3]
  [3 4 5]]
The items in the second column are:
[2 4 5]
The items in the second row are:
[3 4 5]
[[3 4 5]
[4 5 6]]
The items column 1 onwards are:
[[2 3]
 [4 5]
 [5 6]]
```

NUMPY INDEXING AND SLICING

- Multidimensional arrays can have one index per axis.
  - Indices are given in a tuple separated by commas.

```
import numpy as np

myArray = np.array([[0, 1, 2, 3], [10, 11, 12, 13], [20,
21, 22, 23], [30, 31, 32, 33], [40, 41, 42, 43]])

print (myArray)
print ('myArray[2,3]\n', myArray[2,3])

# rows 1:3 in the second column of myArray
print ('myArray[1:3, 1] \n', myArray[1:3, 1])

# each row in the second column of myArray
print ('myArray[ : ,1] \n', myArray[ : ,1])

# each column in the second and third row of myArray
print ('myArray[1:3, : ] \n',myArray[1:3, : ])

# the last row, equivalent to myArray[-1,:]
print ('myArray[-1] \n', myArray[-1])
```

```
[[ 0  1  2  3]
 [10 11 12 13]
 [20 21 22 23]
 [30 31 32 33]
 [40 41 42 43]]

myArray[2,3]
 23

myArray[1:3, 1]
[11 21]

myArray[ : ,1]
 [ 1 11 21 31 41]

myArray[1:3, : ]
 [[10 11 12 13]
 [20 21 22 23]]

myArray[-1]
[40 41 42 43]
```

BOOLEAN INDEXING

```
import numpy as np

myArray = np.array([[ 0, 1, 2],[ 3, 4, 5],[ 6, 7, 8],[ 9, 10, 11]])
print ('printing myArray')
print (myArray)

bools = myArray > 5
print (bools)

# Now we will print the items greater than 5
print ('The items greater than 5 are:')
print (myArray[bools])
```

```
printing myArray
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]

[[False False False]
 [False False False]
 [ True  True  True]
 [ True  True  True]]

The items greater than 5 are:
[ 6  7  8  9 10 11]
```

NUMPY BASIC OPERATIONS

ARITHMETIC OPERATORS

- Arithmetic operators on arrays apply element-wise.
  - A new array is created and filled with the result.

```
import numpy as np

arrayA = np.array( [20,30,40,50] )
arrayB = np.arange( 4 )
print(arrayB)
arrayC = arrayA - arrayB
print(arrayC)
print(arrayB ** 2)
print(10 * np.sin(arrayA))
print( arrayA < 35)
```

```
[0 1 2 3]

[20 29 38 47]

[0 1 4 9]

[ 9.12945251 -9.88031624  7.4511316  -2.62374854]

[ True  True False False]
```

ADDITION and MULTIPLICATION

- The operators *,+ operate element-wise in NumPy arrays.
    - The matrix product can be performed using the dot function.

```
import numpy as np

arrayA = np.array( [[1,1],[0,1]] )
arrayB = np.array( [[2,0],[3,4]] )

print(arrayA * arrayB)        # element-wise product
print(arrayA.dot(arrayB))     # matrix product
print(np.dot(arrayA, arrayB)) # another matrix product
print(arrayA + arrayB)
```

```
[[2 0]
 [0 4]]

[[5 4]
 [3 4]]

[[5 4]
 [3 4]]

[[3 1]
 [3 5]]
```

IN PLACE OPERATIONS

- += and *= operators act in place to modify an existing array rather than create a new one.

```
import numpy as np

arrayA = np.ones((2,3), dtype=int)
arrayB = np.random.random((2,3))
arrayA *= 3
print(arrayA)
print(arrayB)
arrayB += arrayA
print(arrayB)

arrayA += arrayB
```

```
[[3 3 3]
[3 3 3]]

[[0.8714065 0.2653988 0.92162714]
 [0.5503744 0.02992026 0.32266732]]

[[3.8714065 3.2653988 3.92162714]
[3.5503744 3.02992026 3.32266732]]
```

Traceback (most recent call last): File “test.py", line 9, in <module>

typeTypeError: Cannot cast ufunc add output from dtype('float64') to dtype('int64') with casting rule 'same_kind'

UNARY OPERATIONS

- Unary operations, such as computing the sum of all the elements in the array, are implemented as methods of the ndarray class.

```
import numpy as np

myArray = np.random.random((2,3))
print(myArray)
print(myArray.sum())
print(myArray.min())
print(myArray.max())
```

```
[[0.11984434 0.37631986 0.37201106]
 [0.76904911 0.15273116 0.78021234]]

2.570167872017941

0.1198443431329893

0.7802123438932964
```

AXIS SUM

- By specifying the axis parameter you can apply an operation along the specified axis of an array.

```
import numpy as np

myArray = np.arange(12).reshape(3,4)
print(myArray)

# sum of each column
print(myArray.sum(axis=0))

# min of each row
print(myArray.min(axis=1))

# cumulative sum along each row
print(myArray.cumsum(axis=1))
```

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

[12 15 18 21]

[0 4 8]

[[ 0  1  3  6]
 [ 4  9 15 22]
 [ 8 17 27 38]]
```

UNIVERSAL FUNCTIONS

- NumPy provides familiar mathematical functions such as sin, cos, and exp.
  - In NumPy, these are called “universal functions” (ufunc)

```
import numpy as np

arrayA = np.arange(3)
print(arrayA)
print(np.exp(arrayA))
print(np.sqrt(arrayA))
arrayB = np.array([2., -1., 4.])
print(np.add(arrayA, arrayB))
```

```
[0 1 2]
[1.         2.71828183 7.3890561 ]
[0.         1.         1.41421356]
[2. 0. 6.]
```

NUMPY ARRAY MANIPULATION

TRANSPOSE

```
arrayA = np.arange(0,60,5)
arrayA = arrayA.reshape(3,4)
print('printing arrayA')
print(arrayA)

print ('Transpose of arrayA')
arrayB = arrayA.T
print (arrayB)
```

```
printing arrayA
[[ 0  5 10 15]
 [20 25 30 35]
 [40 45 50 55]]

Transpose of arrayA
[[ 0 20 40]
[ 5 25 45]
[10 30 50]
[15 35 55]]
```

CONCATENATION

- concatenation is used to join two or more arrays of the same shape along a specified axis:

```
import numpy as np

arrayA = np.array([[1,2],[3,4]])
print ('printing arrayA')
print (arrayA)

arrayB = np.array([[5,6],[7,8]])
print ('printing arrayB')
print (arrayA)

# both the arrays are of same dimensions
print ('Joining arrayA and arrayB along axis 0:')
print (np.concatenate((arrayA,arrayB)))

print ('Joining arrayA and arrayB along axis 1:')
print (np.concatenate((arrayA,arrayB),axis = 1))
```

```
printing arrayA
[[1 2]
[3 4]]

printing arrayB
[[1 2]
[3 4]]

Joining arrayA and arrayB along axis 0:
[[1 2]
 [3 4]
 [5 6]
 [7 8]]

Joining arrayA and arrayB along axis 1:
[[1 2 5 6]
 [3 4 7 8]]
```

APPEND

```
import numpy as np

myArray = np.array([[1,2,3],[4,5,6]])
print ('printing myArray:')
print (myArray)

print ('Append elements to myArray:')
print (np.append(myArray, [7,8,9]))

print ('Append elements along axis 0:')
print (np.append(myArray, [[7,8,9]],axis = 0))

print ('Append elements along axis 1:')
print (np.append(myArray, [[5,5,5],[7,8,9]],axis = 1))
```

```
printing myArray:
[[1 2 3]
 [4 5 6]]

Append elements to myArray:
[1 2 3 4 5 6 7 8 9]

Append elements along axis 0:
[[1 2 3]
 [4 5 6]
 [7 8 9]]

Append elements along axis 1:
[[1 2 3 5 5 5]
[4 5 6 7 8 9]]
```

INSERT

```
import numpy as np

myArray = np.array([[1,2],[3,4],[5,6]])
print ('printing myArray:')
print (myArray)

print ('Axis parameter not passed. The input array is flattened
before insertion.')
print (np.insert(myArray,3,[11,12]))

print ('Axis parameter passed.')
print (np.insert(myArray,1,[11,12],axis = 0))

print (np.insert(myArray,1,[11,12,13],axis = 1))
```

```
printing myArray:
[[1 2]
 [3 4]
 [5 6]]

Axis parameter not passed. The input array is flattened before insertion.

[ 1  2  3 11 12  4  5  6]

Axis parameter passed.

[[ 1  2]
 [11 12]
 [ 3  4]
 [ 5  6]]

[[ 1 11  2]
 [ 3 12  4]
 [ 5 13  6]]
```

DELETE

- Numpy.delete(arr, obj, axis)

```
import numpy as np

myArray = np.arange(12).reshape(3,4)+10
print ('printing myArray:')
print (myArray)

print ('Array flattened before delete operation as axis not used:')
print (np.delete(myArray,5))

print ('Column 2 deleted:')
print (np.delete(myArray,1,axis = 1))
```

```
printing myArray:
[[10 11 12 13]
 [14 15 16 17]
 [18 19 20 21]]

Array flattened before delete operation as axis not used:
[10 11 12 13 14 16 17 18 19 20 21]

Column 2 deleted:
[[10 12 13]
 [14 16 17]
 [18 20 21]]
```

round, floor, ceil

```
import numpy as np

myArray = np.array([1.0,5.55, 123, 0.567, 25.532])
print ('printing myArray:')
print (myArray)

print ('After rounding:')
print (np.around(myArray))
print (np.around(myArray, decimals = 1))
print (np.around(myArray, decimals = -1))

print ('The floor array:')
print (np.floor(myArray))

print ('The ceil array:')
print (np.ceil(myArray))
```

```
printing myArray:
[  1.      5.55  123.      0.567  25.532]

After rounding:
[  1.   6. 123.   1.  26.]
[  1.    5.6 123.    0.6  25.5]
[  0.  10. 120.   0.  30.]

The floor array:
[  1.   5. 123.   0.  25.]

The ceil array:
[  1.   6. 123.   1.  26.]
```

STATISTICAL FUNCTIONS

MEDIAN

```
import numpy as np

myArray = np.array([[30,65,70],[80,95,10],[50,90,60]])

print ('printing myArray')
print (myArray)

print ('Applying median function:')
print (np.median(myArray))

print ('Applying median function along axis 0:')
print (np.median(myArray, axis = 0))

print ('Applying median function along axis 1:')
print (np.median(myArray, axis = 1))
```

```
printing myArray
[[30 65 70]
 [80 95 10]
 [50 90 60]]

Applying median function:
65.0

Applying median function along axis 0:
[50. 90. 60.]

Applying median function along axis 1:
[65. 80. 60.]
```

MEAN and STANDARD DEVIATION

```
import numpy as np

myArray = np.array([[1,2,3],[3,4,5],[4,5,6]])
print ('printing myArray')
print (myArray)

print ('Applying mean function:')
print (np.mean(myArray))

print ('Applying mean function along axis 0:')
print (np.mean(myArray, axis = 0))

print ('Applying mean function along axis 1:')
print (np.mean(myArray, axis = 1))
print ('Standard Deviation')
print (np.std(myArray))
```

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

Applying mean function:
3.6666666666666665

Applying mean function along axis 0:
[2.66666667 3.66666667 4.66666667]

Applying mean function along axis 1:
[2. 4. 5.]

Standard Deviation
1.4907119849998598
```

THANK YOU!