# Python
# WEEK 8
## Numpy

AI Academy

# COMPUTER PROGRAMMING WITH PYTHON

**Instructor - James E. Robinson, III**

**Teaching Assistant - Travis Martin**

# LIGHTNING REVIEW

- Variables
- Input / Output
- Expressions
- Functions
- Conditional Control
- Looping
- Data Types
- Logging
- Functions
- Scope
- Recursion
- Decorator
- Recursion
- Dynamic Prg
- Exceptions
- Classes
- Objects
- Encapsulation
- Public v/s Private
- Dunder Methods
- Instances
- Inheritance
- Types of Inheritance
- Polymorphism
- Method Overriding
- Queue
- Stacks
- Graphs
- Trees
- Binary Trees
- Traversal Methods
- Searching
- Files
- Opening / Closing
- Reading / Writing
- Context Managers
- File Exceptions
- Handling Exceptions
- File Formats

# TOPICS COVERED

- Numpy
- The ndarray
  - Creation
  - Attributes
  - Data Types
  - Operations
  - Statistics
  - Axis
  - Indexing / Slicing
  - Views / Copies
  - Modification

*NEW MODULE INTRODUCTION*

# NUMPY

# NUMPY

***PYTHON LIBRARY USED FOR WORKING WITH ARRAYS***

**Lists v/s Arrays**

In Python we have lists that serve the purpose of arrays, but they are slow to process.

Unlike lists, NumPy arrays are stored at one continuous location in memory. This allows NumPy functions to access and manipulate them very efficiently.

- Arrays need to be declared. Lists don't
- Arrays can store data very compactly
- Arrays are great for numerical operations

# NDARRAY

***N-DIMENSIONAL ARRAY***

The type() of array in numpy is numpy.ndarray

**Characteristics**

All elements have the same type.

If not specified, then the type will be determined as the minimum type required to hold the objects in the sequence.

Number of dimensions and items in an array is defined by its shape, which is a tuple of N non-negative integers that specify the sizes of each dimension.

eg. (2,3,5) [2 X 3 X 5 grid of array]

# NDARRAY - CREATION

***BY IMPORTING NUMPY MODULE***

**Example**

```
import numpy
x = numpy.array([[1,2,3,4,5],[6,7,8,9,0]])
print(x)
print(type(x))
```

**Alias Example**

```
import numpy as np
x = np.array([[1,2,3,4,5],[6,7,8,9,0]])
print(x)
print(type(x))
```

**OUTPUT**

```
[[1 2 3 4 5]
 [6 7 8 9 0]]
<class 'numpy.ndarray'>
```

**Note**

The lack of ‘ , ’ in the output represents that it is an array and not a list of list

# NDARRAY - ATTRIBUTES

***FUNCTIONS THAT SHOW THE META VALUES OF THE ARRAY***

**dtype**
Data type of items
**size**
Number of items
**itemsize**
Length of one array item in bytes
**ndim**
Dimension,number of indices
**shape**
Number of items in each dimension

**Example**

```
import numpy as np
x = np.array([[1,2,3,4,5],
              [6,7,8,9,0],
              [5,4,3,2,1]])
print(x)
print("Type -",type(x))
print("Data Type -",x.dtype)
print("Size -",x.size)
print("ItemSize -",x.itemsize)
print("Dimensions -",x.ndim)
print("Shape -",x.shape)
```

**OUTPUT**

```
[[1 2 3 4 5]
 [6 7 8 9 0]
 [5 4 3 2 1]]
Type - <class 'numpy.ndarray'>
Data Type - int32
Size - 15
ItemSize - 4
Dimensions - 2
Shape - (3, 5)
```

# NDARRAY - DATA TYPES

***ARRAY CAN HAVE ELEMENTS OF CERTAIN DATA TYPES***

- i - integer
- b - boolean
- u - unsigned integer
- f - float
- c - complex float
- m - timedelta
- M - datetime
- O - object
- S - string
- U - unicode string
- V - fixed chunk of memory for other type ( void )

# NDARRAY - OPERATIONS

***FUNCTIONS THAT CREATE OR INTERACT WITH ARRAYS***

**full**
Fill array with specified value
**zeros**
Fill array with 0s
**ones**
Fill array with 1s
**arange**
Evenly spaced values within a given interval (int)
**linspace**
Evenly spaced numbers over a specified interval (float)

**Example**

```
import numpy as np
x = np.full(5,3)
print("Full:\n",x)
x = np.zeros((2,4))
print("Zeros:\n",x)
x = np.ones((2,5))
print("Ones:\n",x)
x = np.arange(1,7)
print("Range:\n",x)
x = np.linspace(1,5,6)
print("LinSpace:\n",x)
```

**OUTPUT**

```
Full:
 [3 3 3 3 3]
Zeros:
 [[0. 0. 0. 0.]
 [0. 0. 0. 0.]]
Ones:
 [[1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]]
Range:
 [1 2 3 4 5 6]
LinSpace:
 [1.  1.8 2.6 3.4 4.2 5. ]
```

# NDARRAY - OPERATIONS

***SCALAR AND ARRAY OPERATIONS CAN ALSO BE PERFORMED***

**Scalar Operations**
Modifying an array with scalar operations such as *,-,/,+ etc affects the whole array

**Array Operations**
Applying *,-,/,+ etc operations of 2 arrays if and only if their shapes are same

**Example**

```
import numpy as np
x = np.arange(1,10)
print(x)
print(x*2)
print(x-2)
y = np.arange(3,12)
print(x.shape,y.shape)
print(x+y)
print(x*y)
```

**OUTPUT**

```
[1 2 3 4 5 6 7 8 9]
[ 2  4  6  8 10 12 14 16 18]
[-1  0  1  2  3  4  5  6  7]
(9,) (9,)
[ 4  6  8 10 12 14 16 18 20]
[ 3  8 15 24 35 48 63 80 99]
```

# NDARRAY - STATISTICS

## *STATISTIC FUNCTIONS*

**mean**
Finds mean of array
**median**
Finds median of array
**sum**
Calculates sum of array
**std**
Finds standard deviation of array
**var**
Finds variance of array

[More functions]

**Example**

```
import numpy as np
x = np.linspace(1,5,6)
print(x)
print("Mean -")
print(x.mean())
print("Median -")
print(np.median(x))
print("Sum -")
print(np.sum(x))
print("Standard Deviation -")
print(np.std(x))
print("Variance -")
print(np.var(x))
```

**OUTPUT**

```
[1.  1.8 2.6 3.4 4.2 5. ]
Mean -
3.0
Median -
3.0
Sum -
18.0
Standard Deviation -
1.3662601021279464
Variance -
1.8666666666666665
```

# NDARRAY - AXIS

***OPERATIONS ON ARRAYS CAN BE PERFORMED ON SPECIFIC DIMENSIONS***

**Row**
axis = 0

**Column**
axis = 1

**Note:**
If array goes to higher dimensions, those can also be specified.

**Example**

```
import numpy as np
x = np.linspace(1,10,25)
a = x.reshape(5,5)
print(a)
print("Mean -")
print(a.mean())
print("Mean by Rows-")
print(a.mean(axis=0))
print("Mean by Column-")
print(a.mean(axis=1))
```

**OUTPUT**

```
[[ 1.    1.375  1.75   2.125  2.5  ]
 [ 2.875  3.25   3.625  4.     4.375]
 [ 4.75   5.125  5.5    5.875  6.25 ]
 [ 6.625  7.     7.375  7.75   8.125]
 [ 8.5    8.875  9.25   9.625 10.   ]]
Mean -
5.5
Mean by Rows-
[4.75  5.125 5.5   5.875 6.25 ]
Mean by Column-
[1.75  3.625 5.5   7.375 9.25 ]
```

# NDARRAY - INDEXING/SLICING

*NDARRY SUPPORTS MULTIDIMENSIONAL INDEXING*

**Example Array (a)**

```
[[ 1.000 1.375  1.750  2.125  2.500 ]
 [ 2.875  3.250  3.625  4.000  4.375 ]
 [ 4.750  5.125  5.500  5.875  6.250 ]
 [ 6.625  7.000  7.375  7.750  8.125 ]
 [ 8.500  8.875  9.250  9.625  10.00 ]]
```

**Single item at index - *array[row,col]***

*a[2,3]*

```
5.875
```

**Slice of rows - *array[row1:rowN]***

*a[3:4]*

```
[[6.625, 7.   , 7.375, 7.75 , 8.125]]
```

**Specific row - *array[[rowX,rowY,rowN]]***

*a[[3,4,1]]*

```
[[ 6.625,  7.   ,  7.375,  7.75 ,  8.125],
 [ 8.5  ,  8.875,  9.25 ,  9.625, 10.   ],
 [ 2.875,  3.25 ,  3.625,  4.   ,  4.375]]
```

# NDARRAY - INDEXING/SLICING

## *NDARRY SUPPORTS MULTIDIMENSIONAL INDEXING*

**Example Array (a)**

```
[[ 1.000 1.375  1.750  2.125  2.500 ]
 [ 2.875  3.250  3.625  4.000  4.375 ]
 [ 4.750  5.125  5.500  5.875  6.250 ]
 [ 6.625  7.000  7.375  7.750  8.125 ]
 [ 8.500  8.875  9.250  9.625  10.00 ]]
```

**Slice of columns - *array[:,col1:rowN]***

*a[:,1:2]*

```
[[1.375],
 [3.25 ],
 [5.125],
 [7.   ],
 [8.875]]
```

**Specific column - *array[:,[rowX,rowY,rowN]]***

*a[:,[3,1]]*

```
[[2.125, 1.375],
 [4.   , 3.25 ],
 [5.875, 5.125],
 [7.75 , 7.   ],
 [9.625, 8.875]]
```

# NDARRAY - INDEXING/SLICING

## *NDARRY SUPPORTS MULTIDIMENSIONAL INDEXING*

**Example Array (a)**

```
[[ 1.000 1.375  1.750 2.125 2.500 ]
 [ 2.875 3.250  3.625 4.000 4.375 ]
 [ 4.750 5.125  5.500 5.875 6.250 ]
 [ 6.625 7.000  7.375 7.750 8.125 ]
 [ 8.500 8.875  9.250 9.625 10.00 ]]
```

**Slice of columns and rows - *array[row1:rowN:,col1:colN]***

*a[2:4,1:3]*

```
[[5.125, 5.5  ],
 [7.   , 7.375]]
```

**Specific row & column - *array[[rowX,rowY,rowN],[colX,colY,colN]]***

*a[[2,4],[4,1]]*

```
[6.250, 8.875]
```

**Note:**

If ‘ : ’ is given, it returns all rows / columns

Slicing only works for continuous segments while specific indexing can change order as well as can be non-contiguous

# NDARRAY - VIEWS / COPIES

***KNOWING IF YOU ARE OPERATING ON A VIEW OR A COPY IS IMPORTANT***

**View**

A New array object that points to the original arrays data. While it is a different object, the data being operated is same.

**Copy**

A New array object with its own copy of the data
Created by .copy()

**Example**

```
arr = np.array([1,2,3,4,5])
print("Original Before -")
print(arr)
x = arr.view()
x[0] = 9
print("Original for view -")
print(arr)
print("View -")
print(x)
y = arr.copy()
y[1] = 8
print("Original for copy -")
print(arr)
print("Copy -")
print(y)
```

**OUTPUT**

```
Original Before -
[1 2 3 4 5]
Original for view -
[9 2 3 4 5]
View -
[9 2 3 4 5]
Original for copy -
[9 2 3 4 5]
Copy -
[9 8 3 4 5]
```

# NDARRAY - MODIFICATIONS

## *NDARRAYS CAN BE MODIFIED WITHIN REASON*

**reshape**
View of array in a different shape
**resize**
Changes the shape of the original array
**Note:**
New shape must be able to occupy all elements
**flatten**
Copy of array in 1d space
**ravel**
View of array in 1d space
**transpose**
View of array as transpose of original

**Example**

```
import numpy as np
arr = np.array([1,2,3,4,5,6])
arr.resize(2,3)
print(arr)
x = arr.reshape(3,2)
print(x)
y = x.flatten()
z = x.ravel()
y[0] = 99
z[0] = 88
print(x)
print(y)
print(z)
print(x.transpose())
```

**OUTPUT**

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

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

[[88  2]
 [ 3  4]
 [ 5  6]]

[99  2  3  4  5  6]

[88  2  3  4  5  6]

[[88  3  5]
 [ 2  4  6]]
```

# WEEK SUMMARY

- Learned the module Numpy
- Learned concept of arrays
- Understood data types in arrays
- Learned basic and statistical operations of arrays
- Learned to index and slice arrays
- Understood difference between views and copies
- Learned how to modify arrays using numpy

# THANK YOU

**FOR ADDITIONAL QUERIES OR DOUBTS**
**CONTACT:**
**jerobins@ncsu.edu**

**AI** Academy