**Numpy**

**Introduction**

* Third party package developed using Python
* Used for mathematical and statistical calculation
* The earlier name of this package: Numeric
* Using numpy many modules have been developed like
  + Scipy
  + Pandas
  + Sklearn
* Pros
  + Memory efficient (takes very less memory than python collections)
  + Gives way better performance than the python collections
  + Has many operations on matrices / arrays in built
  + Free and open source

**Install numpy**

* pip install numpy
* pip3 install numpy
* Anaconda

**Data types**

* Numbers
  + Whole number
    - Signed integers
      * E.g. -10
      * int
      * int8
      * int16
      * int32
      * int64
    - Unsigned integers
      * E.g. 10
      * uint
      * uint8
      * uint16
      * uint32
      * uint64
  + Decimal number
    - E.g. 5.6
    - float
    - float16
    - float32
    - float64
    - float128
* String
  + E.g. “test”
  + string\_
* Boolean
  + E.g. True / False
  + bool\_
* Complex
  + E.g. (5 + 3j)
  + complex
  + complex8
  + complex16
  + complex32
  + complex64
  + complex128

**Special values**

* NaN
  + Not a Number
* Inf
  + Infinity

**Arrays**

* Collection of similar values
* ndarray (n-dimensional array) is used to create an array
* the memory gets allocated contiguously
* once created, the size of array CAN NOT be changed [immutable]
* numpy array is ALWAYS way faster than python collection
* Create
  + array()
    - syntax:
      * np.array(<list>/<items>)
      * np.array(<list>/<items>, dtype=<type>)
    - e.g.
      * np.array([1, 2, 3, 4]) # [1 2 3 4] -> int64
      * np.array([1, 2, 3], dtype=np.float) # [1. 2. 3.] -> float
      * np.array([‘test5’], dtype=’<U2’) # [‘te’] -> <U2
  + arange()
    - used to create an array with start and stop values
    - e.g.
      * a1 = np.array(np.arange(5)) # [0 1 2 3 4]
  + zeros()
    - used to generate an array pre-filled with zeros
    - e.g.
      * a1 = np.zeros(5) # [0 0 0 0 0]
      * a1 = np.zeros((2,2)) # [ [ 0 0 ] [ 0 0 ]]
  + ones()
    - used to generate an array pre-filled with one
    - e.g.
      * a1 = np.ones(5) # [1 1 1 1 1]
      * a1 = np.ones((2,2)) # [ [ 1 1 ] [ 1 1 ]]
  + empty()
    - used to generate an array pre-filled with zeros
    - e.g.
      * a1 = np.zeros(5) # [0 0 0 0 0]
      * a1 = np.zeros((2,2)) # [ [ 0 0 ] [ 0 0 ]]
  + ndarray()
    - syntax:
      * np.ndarray(<list>/<items>)
      * np.ndarray(<list>/<items>, dtype=<type>)
    - e.g.
      * np.array([1, 2, 3, 4]) # [1 2 3 4] -> int64
      * np.array([1, 2, 3], dtype=np.float) # [1. 2. 3.] -> float
  + np.array([‘test5’], dtype=’<U2’) # [‘te’] -> <U2
  + np.random.random()
    - used to generate random numbers
    - e.g.
      * a1 = np.random.randn(3) # [ -1.3 1.4 1.6]
  + np.random.randint()
    - used to generate random integers
    - e.g.
      * a1 = np.random.randint(3, size = 2) # [ 1 2 ]
  + copy()
    - used to create a “new” copy of existing object keeping the same values in newly created object
    - e.g.
      * a1 = np.array([ 1, 2, 3, 4, 5])
      * a2 = a1
      * a1[2] = 5 # a1[2] = 5, a2[2] = 5
      * a3 = a1.copy()
      * a1[2] = 5 # a1[2] = 5, a2[2] = 3
* Attributes
  + shape
    - number of cols and rows in the array
    - e.g.
      * a1 = np.array([1, 2, 3]) # (3,)
      * a2 = np.array([[1, 2], [3, 4]]) # (2, 2)
  + ndim
    - returns the number of dimensions present in the array
    - e.g.
      * a1 = np.array([1, 2, 3]) # 1
      * a2 = np.array([[1, 2], [3, 4]]) # 2
  + itemsize
    - returns size of single item in the array
    - e.g.
      * a1 = np.array([1, 2, 3]) # 8
  + dtype
    - Returns the data type of the array member
    - E.g.
      * a1 = np.array([1, 2, 3, 4]) # int64
      * a2 = np.array([‘test’, ‘test1’]) # string\_ (<U5)
* Slicing
  + Please look into python notes for further details
  + E.g.
    - a1 = np.array([11, 23, 46, 67, 89])
    - a1[0] # 11
    - a1[0:3] # [11 23 46]
    - a1[2:] # [46 67 89]
    - a1[:2] # [11 23]
    - a1[a1 > 50] # [67 89]
    - a1[a1 < 40] # [11 23]
* Mathematical functions
  + Mathematical operations are possible on array
    - E.g.
      * a1 = np.array([1, 2, 3, 4, 4])
      * a2 = np.array([5, 6, 7, 8, 9])
      * a3 = a1 + a2 # [6 8 10 12 12]
  + Broadcasting operations
    - Operation performed with every item present in the array
    - E.g.
      * a1 = np.array([1, 2, 3, 4, 5])
      * Arithmetic
        + Multiplication:

a1 \* 10 # [10 20 30 40 50]

* + - * + Addition

a1 + 20 # [21 22 23 24 25]

* + - * + Subtraction

a1 – 2 # [-1 0 1 2 3]

* + - * + Division

a1 / 3 # [0.3 2/3 1 4/3 5/3]

* + - * Logical
        + less than

a1 < 3 # [True True False False False]

* + - * + greater than

a1 > 2 # [False False True True True]

* + - * + >=
        + <=
        + !=
* Statistical operations
  + mean()
    - used to get mean of an array
    - e.g.
      * a1 = np.array([20, 10, 20, 30, 40, 20, 10, 40, 50, 60, 70])
      * print(a1.mean()) # 33.63636363636363
  + median()
    - used to get median
    - e.g.
      * print(np.median(a1)) # 30.0
  + std()
    - used to calculate standard deviation
    - e.g.
      * print(np.std(a1)) # 19.198829165402614
  + var()
    - used to calculate variance
    - e.g.
      * print(np.var(a1)) # 368.5950413223141
  + mode()
    - used to get mode of an array
    - e.g.
      * from scipy import stats
      * print(stats.mode(a1)) # mode = 20, count = 3
* Methods
  + reshape():
    - used to change the shape of an array
    - e.g.
      * a1 = np.array([1, 2, 3, 4]) # [1 2 3 4]
      * a2 = a1.reshape((2, 2)) # [ [ 1 2 ] [3 4] ]
* Functions
  + max(): used to get maximum value from an array
  + min(): used to get minimum value from an array
  + len(): used to get the length of an array