# NumPy

# Introduction

- Stands for numerical python.
- Fundamental Package for numerical Computations.
- Supports N dimensional array objects that can be used for processing multi-dimensional data.
- Supports different data types.

# Introduction

- Using Numpy we can perform
  - Mathematical and logical operations on arrays.
  - Fourier Transform
  - Linear Algebra Operations
  - Random Number generations

# Crate an array

- Array is the ordered collection of basic data types of given length
- Syntax: numpy.array(object)

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

In [2]: x = np.array([2,3,4,8])

In [3]: type(x)
Out[3]: numpy.ndarray

In [4]: x
Out[4]: array([2, 3, 4, 8])
```

# Array

- Numpy can handle different categorical entries

```
In [5]: y = np.array([2,'LPU',4,8])

In [6]: y

Out[6]: array(['2', 'LPU', '4', '8'], dtype='<U11')
```

# Generate Array using linspace

- Return equaly spaced numbers within the given range.
- numpy.linspace(start, stop, num, dtype, retstep, dtype)
  - Start : start of interval range
  - Stop: The end value of the interval range
  - Num: number of samples to be generated
  - dtype: type of output array
  - restep: return the samples, step value

# Generate Array using linspace

- Generate an array b with start value 1 and stop value 5.

```
In [9]: b = np.linspace(1, 5, num=10, endpoint = True, retstep=False)

In [10]: b

Out[10]: array([1.        , 1.44444444, 1.88888889, 2.33333333, 2.77777778,
       3.22222222, 3.66666667, 4.11111111, 4.55555556, 5.        ])

In [17]: b = np.linspace(1, 5, num=10, endpoint = True, retstep=True)

In [18]: b

Out[18]: (array([1.        , 1.44444444, 1.88888889, 2.33333333, 2.77777778,
        3.22222222, 3.66666667, 4.11111111, 4.55555556, 5.        ]),
 0.4444444444444444)

In [19]: b = np.linspace(1, 5, num=10, endpoint = False, retstep=True)

In [20]: b

Out[20]: (array([1. , 1.4, 1.8, 2.2, 2.6, 3. , 3.4, 3.8, 4.2, 4.6]), 0.4)
```

# Generate array using np.arange

- np.arange returns equally spaced numbers within the given range based on step size.
- Syntax: np.arange(start, stop, stepsize)
  - Start : start of interval range
  - Stop: The end value of the interval range
  - Stepsize: stepsize in the interval

# Generate array using np.arange

- Generate an array with start value = 1, stop value = 10 and stepsize = 2

```
In [23]: d = np.arange(1,10,2)

In [24]: d

Out[24]: array([1, 3, 5, 7, 9])
```

# Generate array using ones

- Numpy.ones returns an array of given shape and type filled with ones.
- Syntax: numpy.ones(shape, dtype)
- Shape: integer or sequence of integers
- Dtype: data type (default: float)

```
In [27]: e = np.ones((3,4))

In [28]: e

Out[28]: array([[1., 1., 1., 1.],
                [1., 1., 1., 1.],
                [1., 1., 1., 1.]])
```

# Generate array using zeros

- Numpy.zeros returns an array of given shape and type filled with zeros.
- Syntax: numpy.zeros(shape, dtype)
- Shape: integer or sequence of integers
- Dtype: data type (default: float)

```
f = np.zeros((3,4))
```

```
f
```

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

# Generate array using random.rand()

- Numpy.random.rand() returns an array of given shape filled with random values.
- Syntax: numpy.random.rand(shape)

```
g = np.random.rand(5)
```

```
g
```

```
array([0.96591181, 0.17853905, 0.82199814, 0.08965516, 0.32435735])
```

```
In [43]: g = np.random.rand(5,4)

In [44]: g

Out[44]: array([[0.18836923, 0.25885422, 0.3509376 , 0.46863139],
                [0.21263104, 0.18189903, 0.1027898 , 0.2155268 ],
                [0.68860479, 0.49224318, 0.00386144, 0.41830482],
                [0.73607213, 0.06082685, 0.86523608, 0.13471038],
                [0.77180107, 0.43262638, 0.68347156, 0.1230697 ]])
```

# Advantages of numpy

- Numpy supports vectorized operations.
- Array operations are carried out in c and hence universal functions in numpy are faster than operations carried out in Python Lists.

```
In [68]: x = range(1000)

In [69]: timeit(sum(x))
         37.6 µs ± 4.86 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)

In [70]: y = np.array(x)

In [ ]: y

In [72]: timeit(np.sum(y))
         9.93 µs ± 580 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

# Advanages of numpy: Consumes less storage space

```
In [73]: import sys

In [87]: sys.getsizeof(1) * len(x)
Out[87]: 28000

In [90]: y.itemsize * y.size
Out[90]: 4000
```

# Reshape an array

- Recast an array to new shape without changing its data

```
In [41]: g = np.arange(1,10).reshape(3,3)

In [42]: g

Out[42]: array([[1, 2, 3],
                [4, 5, 6],
                [7, 8, 9]])
```

# Array Dimensions

- Shape Returns the dimensions of an array

```
In [43]: h = np.array([[1,4,8],[2,6,9],[3,5,7]])

In [44]: h

Out[44]: array([[1, 4, 8],
                [2, 6, 9],
                [3, 5, 7]])

In [46]: h.shape

Out[46]: (3, 3)
```

# Numpy addition

- Numpy.add() returns the element wise addition between two arrays.

```
In [47]: g

Out[47]: array([[1, 2, 3],
                [4, 5, 6],
                [7, 8, 9]])

In [48]: h

Out[48]: array([[1, 4, 8],
                [2, 6, 9],
                [3, 5, 7]])

In [50]: np.add(g,h)

Out[50]: array([[ 2,  6, 11],
                [ 6, 11, 15],
                [10, 13, 16]])
```

# Numpy Multiplication

- Numpy.multiply(): returns element wise multiplication between two arrays.

```
Out[47]: array([[1, 2, 3],
                [4, 5, 6],
                [7, 8, 9]])

In [48]: h

Out[48]: array([[1, 4, 8],
                [2, 6, 9],
                [3, 5, 7]])

In [51]: np.multiply(g,h)

Out[51]: array([[ 1,  8, 24],
                [ 8, 30, 54],
                [21, 40, 63]])
```

# Other numpy functions

| Function name | Description |
|---|---|
| numpy.subtract | performs element wise subtraction between two arrays |
| numpy.divide | returns an element wise division of inputs |
| numpy.remainder | Return element-wise remainder of division |

## Accessing Components of an array

- Components of an array can be accessed using index number.

| | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 1 | 2 | 3 |
| 1 | 4 | 5 | 6 |
| 2 | 7 | 8 | 9 |

- Extract element 2 in array a : a(0,1)

```
In [55]: h
Out[55]: array([[1, 4, 8],
                [2, 6, 9],
                [3, 5, 7]])

In [57]: h[0,1]
Out[57]: 4

In [60]: h[1:3]
Out[60]: array([[2, 6, 9],
                [3, 5, 7]])

In [61]: h[:,0]
Out[61]: array([1, 2, 3])

In [62]: h
Out[62]: array([[1, 4, 8],
                [2, 6, 9],
                [3, 5, 7]])

In [64]: h[0,:]
Out[64]: array([1, 4, 8])
```

# Modifying an array using transpose

- Numpy.transpose(array): returns the transpose of an array.

```
In [73]: h

Out[73]: array([[ 1, 10,  8],
                [ 2, 20,  9],
                [ 3,  5,  7]])

In [74]: np.transpose(h)

Out[74]: array([[ 1,  2,  3],
                [10, 20,  5],
                [ 8,  9,  7]])
```

# Modify array using append

- Add the values at the end of array.
- Syntax: numpy.append(array,axis)

```
In [99]: h = np.append(h, [[12,34,65]], axis = 0)

In [106]: h

Out[106]: array([[ 1, 10,  8],
                 [ 2, 20,  9],
                 [ 3,  5,  7],
                 [12, 34, 65],
                 [12, 34, 65]])

In [107]: col = np.array([[23,45,67,98,24]]).reshape(5,1)

In [108]: h = np.append(h,col, axis = 1)

In [109]: h

Out[109]: array([[ 1, 10,  8, 23],
                 [ 2, 20,  9, 45],
                 [ 3,  5,  7, 67],
                 [12, 34, 65, 98],
                 [12, 34, 65, 24]])
```

```
Out[109]: array([[ 1, 10,  8, 23],
                 [ 2, 20,  9, 45],
                 [ 3,  5,  7, 67],
                 [12, 34, 65, 98],
                 [12, 34, 65, 24]])
```

```
In [111]: h = np.insert(h, 1, [[10,20,30, 40]], axis = 0)
```

```
In [112]: h
```

```
Out[112]: array([[ 1, 10,  8, 23],
                 [10, 20, 30, 40],
                 [ 2, 20,  9, 45],
                 [ 3,  5,  7, 67],
                 [12, 34, 65, 98],
                 [12, 34, 65, 24]])
```

```
In [114]: h = np.delete(h, 2, axis = 0)
```

```
In [115]: h
```

```
Out[115]: array([[ 1, 10,  8, 23],
                 [10, 20, 30, 40],
                 [ 3,  5,  7, 67],
                 [12, 34, 65, 98],
                 [12, 34, 65, 24]])
```