Python Libraries

- Nympy
- Pandas
- Scikit learn
- Matplotlib
- Seaborn

Numpy

- Numpy is a library written for scientific computing and data analysis.
- Designed to carryout mathematical computations in a easier way

advantages

- Numpy is much faster compare to standard python while doing computations.
-

Instaling numpy

conda install numpy

or

pip install numpy

```
## importing
import numpy as np
```

```
## version
np.__version__
```

```
'1.19.5'
```

```
dir(np)
```

```
 '__builtins__',
 '__cached__',
 '__config__',
 '__dir__',
 '__doc__',
 '__file__',
 '__getattr__',
 '__git_revision__',
 '__loader__',
```

```
'__name__',
'__package__',
'__path__',
'__spec__',
'__version__',
'_add_newdoc_ufunc',
'_distributor_init',
'_globals',
'_mat',
'_pytesttester',
'abs',
'absolute',
'add',
'add_docstring',
'add_newdoc',
'add_newdoc_ufunc',
'alen',
'all',
'allclose',
'alltrue',
'amax',
'amin',
'angle',
'any',
'append',
'apply_along_axis',
'apply_over_axes',
'arange',
'arccos',
'arccosh',
'arcsin',
'arcsinh',
'arctan',
'arctan2',
'arctanh',
'argmax',
'argmin',
'argpartition',
'argsort',
'argwhere',
'around',
'array',
'array2string',
'array_equal',
'array_equiv',
'array_repr',
'array_split',
'array_str',
'asanyarray',
'asarray',
'asarray_chkfinite'
```

Creation of arrays in different dimensions.

- We can create a numpy by using List or tuple

```
### 1-Dimensional array
n1 = np.array([0,1,2,3,4])
n1
```

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

```
print(type(n1))
print(n1.ndim) ## to check no of dimensions
print(n1.itemsize)## memory used by each element in bytes
```

```
<class 'numpy.ndarray'>
1
8
```

```
n1.dtype
```

```
dtype('int64')
```

```
a = np.array([2,30,98.7,6,'string'])
a.dtype
```

```
dtype('<U32')
```

```
## 2-Dimensional array
### np.array([[row1], [row2], ... , [row N]])
n2 = np.array([[1,2,3,4],[10,11,12,13],[20,30,40,50]])
n2
```

```
array([[ 1,  2,  3,  4],
       [10, 11, 12, 13],
       [20, 30, 40, 50]])
```

```
print(n2.ndim)
print(n2.shape)# no of rows and columns
```

```
2
(3, 4)
```

```
## 3-D or multi dimentional array
###np.array([[[row1],[row2], ... ,[row N]]])
n3 = np.array([[[1,2,3],[11,12,13],[21,22,23]]])
n3
```

```
array([[[ 1,  2,  3],
        [11, 12, 13],
        [21, 22, 23]]])
```

```
n3.ndim
```

```
3
```

```
### np.array([[ [],[] ], [ [], [] ]])
n33 = np.array([[ [1,2,3],[4,5,6] ], [ [11,12,13], [14,15,16] ]])
n33
```

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

       [[11, 12, 13],
        [14, 15, 16]]])
```

```
n33.ndim
```

```
3
```

```
np.append(n1,12.1)
```

```
array([ 0. ,  1. ,  2. ,  3. ,  4. , 12.1])
```

```
np.append(n1,['s',22])
```

```
array(['0', '1', '2', '3', '4', 's', '22'], dtype='<U21')
```

- np.arange()
- np.full()
- np.ones()
- np.zeros()
- np.linspace()

np.arange()

```
# for loop vs np.arange()

for i in range(1,11):
  print(i, end = ' ')
```

```
1 2 3 4 5 6 7 8 9 10
```

```
np.arange(1,11)
##      start, stop
```

```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

```
np.arange(0,100,5)
#     start, stop, step size
```

```
array([ 0,  5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80,
       85, 90, 95])
```

np.linspace()

```
np.linspace(0,10,20)
```

```
array([ 0.        ,  0.52631579,  1.05263158,  1.57894737,  2.10526316,
        2.63157895,  3.15789474,  3.68421053,  4.21052632,  4.73684211,
        5.26315789,  5.78947368,  6.31578947,  6.84210526,  7.36842105,
        7.89473684,  8.42105263,  8.94736842,  9.47368421, 10.        ])
```

```
np.linspace(0,10,20, retstep = True)
```

```
(array([ 0.        ,  0.52631579,  1.05263158,  1.57894737,  2.10526316,
         2.63157895,  3.15789474,  3.68421053,  4.21052632,  4.73684211,
         5.26315789,  5.78947368,  6.31578947,  6.84210526,  7.36842105,
         7.89473684,  8.42105263,  8.94736842,  9.47368421, 10.        ]),
 0.5263157894736842)
```

```
2.10526316 - 1.57894737
```

```
0.5263157899999997
```

```
help(np.linspace)
```

```
        between samples.
    dtype : dtype, optional
        The type of the output array.  If `dtype` is not given, infer the data
        type from the other input arguments.

        .. versionadded:: 1.9.0

    axis : int, optional
        The axis in the result to store the samples.  Relevant only if start
        or stop are array-like.  By default (0), the samples will be along a
        new axis inserted at the beginning. Use -1 to get an axis at the end.

        .. versionadded:: 1.16.0

    Returns
    -------
    samples : ndarray
        There are `num` equally spaced samples in the closed interval
        ``[start, stop]`` or the half-open interval ``[start, stop)``
        (depending on whether `endpoint` is True or False).
    step : float, optional
        Only returned if `retstep` is True

        Size of spacing between samples.


    See Also
    --------
    arange : Similar to `linspace`, but uses a step size (instead of the
             number of samples).
    geomspace : Similar to `linspace`, but with numbers spaced evenly on a log
                scale (a geometric progression).
    logspace : Similar to `geomspace`, but with the end points specified as
               logarithms.

    Examples
    --------
```

```
>>> np.linspace(2.0, 3.0, num=5)
array([2.  , 2.25, 2.5 , 2.75, 3.  ])
>>> np.linspace(2.0, 3.0, num=5, endpoint=False)
array([2. ,  2.2,  2.4,  2.6,  2.8])
>>> np.linspace(2.0, 3.0, num=5, retstep=True)
(array([2.  ,  2.25,  2.5 ,  2.75,  3.  ]), 0.25)

Graphical illustration:

>>> import matplotlib.pyplot as plt
>>> N = 8
>>> y = np.zeros(N)
>>> x1 = np.linspace(0, 10, N, endpoint=True)
>>> x2 = np.linspace(0, 10, N, endpoint=False)
>>> plt.plot(x1, y, 'o')
[<matplotlib.lines.Line2D object at 0x...>]
>>> plt.plot(x2, y + 0.5, 'o')
[<matplotlib.lines.Line2D object at 0x...>]
>>> plt.ylim([-0.5, 1])
(-0.5, 1)
>>> plt.show()
```

np.full()

```
np.full()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-32-02539d355f6f> in <module>()
----> 1 np.full()

TypeError: full() missing 2 required positional arguments: 'shape' and 'fill_value'
```

SEARCH STACK OVERFLOW

```
np.full(5,2)
## 5 elements full of 2
```

```
array([2, 2, 2, 2, 2])
```

```
np.full(5,2, dtype = 'float')
```

```
array([2., 2., 2., 2., 2.])
```

```
np.full([3,3],'numpy')
```

```
array([['numpy', 'numpy', 'numpy'],
       ['numpy', 'numpy', 'numpy'],
       ['numpy', 'numpy', 'numpy']], dtype='<U5')
```

np.ones()

```
np.ones(5)
```

```
array([1., 1., 1., 1., 1.])
```

```
np.ones(5,dtype = 'int')
```

```
array([1, 1, 1, 1, 1])
```

```
np.ones([3,3])
```

```
array([[1., 1., 1.],
       [1., 1., 1.],
       [1., 1., 1.]])
```

```
np.ones([3,4,5])
```

```
array([[[1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.]],

       [[1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.]],

       [[1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.]]])
```

```
np.ones([3,4,5])[0]
```

```
array([[1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.]])
```

np.zeros()

- Similar to np.ones, but array contains 0

Task1

- Try to do np.zeros() by yourself

Accessing the elements from the array

```
a = np.arange(10,21)
a
```

```
array([10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20])
```

```
a[0]
```

```
10
```

```
a[-1]# -ve indexing
```

```
20
```

```
a[:3] # first 3 values
```

```
array([10, 11, 12])
```

```
a[-3:]# last 3 values
```

```
array([18, 19, 20])
```

```
a[::2]# every 2nd value
```

```
array([10, 12, 14, 16, 18, 20])
```

```
a[::3]# every 3rd value
```

```
array([10, 13, 16, 19])
```

```
a[::-1]# reverse array
```

```
array([20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10])
```

```
a[[3,6]] # multiple values
```

```
array([13, 16])
```

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

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

```
b[0]
```

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

```
b[0][3]
```

```
4
```

```
b[:,-1]
```

```
array([5, 0])
```

```
b[:,::-1]
```

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

✓ 0s completed at 4:26 PM