arrayObject

November 25, 2022

1 Numpy

- Import Numpy
- Mathmatical Operations
- Array Creation
- Adding Dimension
- np.zeros
- np.eye
- np.empty
- np.full
- np.arange / array range
- np.linspace (differnce between arange & linspace)
- random & randomn, randint
- Types of Array
- NDarray, item, itemsize
- Slicing

[193]:
```
import numpy as np
np.__version__
```

[193]: '1.23.3'

2 Math Computational

[194]:
```
summ = 0
for num in range(1, 9999999):
    summ += num**2*3

summ
```

[194]: 99999955000006499997

[195]:
```
np.arange(1,9999999)
```

[195]: array([1,   2,   3, …, 9999996, 9999997, 9999998])

2.1 Comparing Computational efficiency between general & numpy

[196]:
```
# General Python, takes more time
summ1 = 0
for num in range (1,99999999):
    summ1 += num

summ1
```

[196]:
```
4999999850000001
```

[197]:
```
# Numpy sum operation     takes less time, in case of huge data we will see the␣
↪different more clearly
np.sum(np.arange(1,99999999, dtype=np.int64))
```

[197]:
```
4999999850000001
```

[198]:
```
np.arange(99999999, dtype=np.int64).sum()   # alternative syntax of doing sum␣
↪aggregation over numpy
```

[198]:
```
4999999850000001
```

2.1.1 Numpy is more efficinet

- Previous one needs more time than numpy
- for this operation, numpy is not taking for loop, while for same general python using loop

3 Array Creation

[199]:
```
# np.zeros([shape size] comma dtype with np.bit size)  # Syntax
np.zeros([3,3], dtype=np.int64)
```

[199]:
```
array([[0, 0, 0],
       [0, 0, 0],
       [0, 0, 0]], dtype=int64)
```

[200]:
```
np.zeros([5,5], dtype=np.int64)
```

[200]:
```
array([[0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0]], dtype=int64)
```

[201]:
```
np.zeros([4, 4], dtype=np.float64)
```

[201]:
```
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
```

```
        [0., 0., 0., 0.],
        [0., 0., 0., 0.]])
```

[202]:
```
np.zeros([2,2], dtype=np.float16)
```

[202]:
```
array([[0., 0.],
       [0., 0.]], dtype=float16)
```

[203]:
```
np.zeros([4,3], dtype=np.int64)  # rows & columns is 4:3
```

[203]:
```
array([[0, 0, 0],
       [0, 0, 0],
       [0, 0, 0],
       [0, 0, 0]], dtype=int64)
```

[204]:
```
np.zeros([5,3], dtype=np.int64)
```

[204]:
```
array([[0, 0, 0],
       [0, 0, 0],
       [0, 0, 0],
       [0, 0, 0],
       [0, 0, 0]], dtype=int64)
```

3.0.1 Adding Demension

[205]:
```
np.zeros([4,3,2], dtype=np.int64)
# 4 rows, 3 columns and 2 cell/value per row  or for each row, here is adding
 ↪dimenstion as 2
```

[205]:
```
array([[[0, 0],
        [0, 0],
        [0, 0]],

       [[0, 0],
        [0, 0],
        [0, 0]],

       [[0, 0],
        [0, 0],
        [0, 0]],

       [[0, 0],
        [0, 0],
        [0, 0]]], dtype=int64)
```

[206]:
```
np.zeros([4,3,3], dtype=np.int64)    # 3 cell/ value per row
```

[206]:
```
array([[[0, 0, 0],
        [0, 0, 0],
        [0, 0, 0]],

       [[0, 0, 0],
        [0, 0, 0],
        [0, 0, 0]],

       [[0, 0, 0],
        [0, 0, 0],
        [0, 0, 0]],

       [[0, 0, 0],
        [0, 0, 0],
        [0, 0, 0]]], dtype=int64)
```

[207]:
```python
np.zeros([4,3,4], dtype=np.int64)     # 4 cell/ value per row
```

[207]:
```
array([[[0, 0, 0, 0],
        [0, 0, 0, 0],
        [0, 0, 0, 0]],

       [[0, 0, 0, 0],
        [0, 0, 0, 0],
        [0, 0, 0, 0]],

       [[0, 0, 0, 0],
        [0, 0, 0, 0],
        [0, 0, 0, 0]],

       [[0, 0, 0, 0],
        [0, 0, 0, 0],
        [0, 0, 0, 0]]], dtype=int64)
```

[208]:
```python
np.zeros([4,3,5], dtype=np.int64)     # 5 cell/ value per row
```

[208]:
```
array([[[0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0]],

       [[0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0]],

       [[0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0]],
```

```
        [[0, 0, 0, 0, 0],
         [0, 0, 0, 0, 0],
         [0, 0, 0, 0, 0]]], dtype=int64)
```

[209]:
```
np.zeros([4,3], dtype=np.int64)
```

[209]:
```
array([[0, 0, 0],
       [0, 0, 0],
       [0, 0, 0],
       [0, 0, 0]], dtype=int64)
```

[210]:
```
np.zeros([4,5,10], dtype=np.int64)    # 10 dimension
```

[210]:
```
array([[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]],

       [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]],

       [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]],

       [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]], dtype=int64)
```

[211]:
```
np.zeros([4,8,10], dtype=np.int64)
```

[211]:
```
array([[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
```

```
         [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]],

        [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
         [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
         [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
         [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
         [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
         [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
         [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
         [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]],

        [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
         [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
         [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
         [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
         [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
         [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
         [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
         [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]],

        [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
         [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
         [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
         [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
         [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
         [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
         [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
         [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]], dtype=int64)
```

[212]:
```
np.zeros([4,3,10], dtype=np.int64)
```

[212]:
```
array([[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]],

       [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]],

       [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]],

       [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]], dtype=int64)
```

[213]:
```
np.zeros([4,2,10], dtype=np.int64)
```

[213]:
```
array([[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]],

       [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]],

       [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]],

       [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]], dtype=int64)
```

[214]:
```
np.ones([4,4])    # by default dtype takes float & mentioning dtype is optional
```

[214]:
```
array([[1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.]])
```

[215]:
```
np.ones([4,4], dtype=np.float64)
```

[215]:
```
array([[1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.]])
```

[216]:
```
np.ones([4,4], dtype=np.int64)
```

[216]:
```
array([[1, 1, 1, 1],
       [1, 1, 1, 1],
       [1, 1, 1, 1],
       [1, 1, 1, 1]], dtype=int64)
```

[217]:
```
np.eye(3,3)     # np.eye means "Identity Matrix", eye operation will not take␣
 ↪square bracket [], np.eye takes only "tuple"
```

[217]:
```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

[218]:
```
np.eye(3,3, k=1)    # by default k = 0
```

[218]:
```
array([[0., 1., 0.],
       [0., 0., 1.],
       [0., 0., 0.]])
```

[219]:
```python
np.eye(6,6)        # default k = 0
```

[219]:
```
array([[1., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 0., 1.]])
```

[220]:
```python
np.eye(6,6, k=0)
```

[220]:
```
array([[1., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 0., 1.]])
```

[221]:
```python
np.eye(6,6, k=1)
```

[221]:
```
array([[0., 1., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 0., 1.],
       [0., 0., 0., 0., 0., 0.]])
```

[222]:
```python
np.eye(6,6, k=2)
```

[222]:
```
array([[0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 0., 1.],
       [0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0.]])
```

[223]:
```python
np.eye(6,6, k=4)
```

[223]:
```
array([[0., 0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 0., 1.],
       [0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0.]])
```

[224]:
```python
np.eye(6,6, k=-4)
```

[224]:
```
array([[0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0.]])
```

[225]:
```
np.eye(6,6, k=5)
```

[225]:
```
array([[0., 0., 0., 0., 0., 1.],
       [0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0.]])
```

[226]:
```
np.eye(3,3, k=-1)
```

[226]:
```
array([[0., 0., 0.],
       [1., 0., 0.],
       [0., 1., 0.]])
```

[227]:
```
np.eye(3,4)
```

[227]:
```
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.]])
```

[228]:
```
np.eye(3,4,2)
```

[228]:
```
array([[0., 0., 1., 0.],
       [0., 0., 0., 1.],
       [0., 0., 0., 0.]])
```

[229]:
```
np.eye(3,4,-2)
```

[229]:
```
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [1., 0., 0., 0.]])
```

[230]:
```
np.eye(3,4,3)
```

[230]:
```
array([[0., 0., 0., 1.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
```

[231]:
```
np.eye(3,4,4)
```

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

```
[232]: np.empty([3,3])
```

```
[232]: array([[0., 0., 0.],
              [1., 0., 0.],
              [0., 1., 0.]])
```

```
[233]: np.empty([3,3,3])
```

```
[233]: array([[[1.14410197e-311, 8.25089629e-322, 0.00000000e+000],
               [0.00000000e+000, 0.00000000e+000, 1.15998412e-028],
               [4.31603868e-080, 1.94919985e-153, 1.35717430e+131]],

              [[4.57669057e-072, 1.81148490e-152, 6.14099335e-071],
               [1.05132387e-153, 6.01391519e-154, 1.05135742e-153],
               [8.15766128e+140, 6.01347002e-154, 8.01768646e-096]],

              [[5.49891556e-095, 8.78959876e+198, 2.18229349e-094],
               [1.43267083e+161, 2.20832505e-094, 9.80058441e+252],
               [1.23971686e+224, 2.59923248e-306, 0.00000000e+000]]])
```

```
[234]: np.empty([3,3,2])
```

```
[234]: array([[[1.37962049e-306, 1.24610791e-306],
               [1.11260959e-306, 1.69109959e-306],
               [9.34603679e-307, 1.42419802e-306]],

              [[1.78019082e-306, 4.45061456e-308],
               [1.24612081e-306, 1.37962049e-306],
               [9.34597567e-307, 1.29061821e-306]],

              [[1.78019625e-306, 1.11255866e-306],
               [8.90098127e-307, 9.34609790e-307],
               [3.91792279e-317, 2.44771535e-307]]])
```

```
[235]: np.empty([3,2,2])
```

```
[235]: array([[[0., 0.],
               [0., 0.]],

              [[0., 0.],
               [0., 0.]],

              [[0., 0.],
               [0., 0.]]])
```

[236]:
```
np.full([3,3], 8.5)   # it will fill the 3x3 matrix with 8.5
```

[236]:
```
array([[8.5, 8.5, 8.5],
       [8.5, 8.5, 8.5],
       [8.5, 8.5, 8.5]])
```

[237]:
```
np.full([3,3], 10.5)
```

[237]:
```
array([[10.5, 10.5, 10.5],
       [10.5, 10.5, 10.5],
       [10.5, 10.5, 10.5]])
```

3.0.2 Arange

- np.arange will not take squre backet
- mentioning dtype is optional
- it will take (start: end: step size)
- default takes int
- float & int can be mentioned in dtype
- its like a range function
- it can be reshaped

[238]:
```
np.arange(1,100)
```

[238]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
       18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34,
       35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51,
       52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68,
       69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85,
       86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99])
```

[239]:
```
np.arange(1,100, dtype=np.int64)
```

[239]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
       18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34,
       35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51,
       52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68,
       69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85,
       86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99],
      dtype=int64)
```

[240]:
```
np.arange(1,100, dtype=np.float64)
```

[240]:
```
array([ 1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10., 11., 12., 13.,
       14., 15., 16., 17., 18., 19., 20., 21., 22., 23., 24., 25., 26.,
       27., 28., 29., 30., 31., 32., 33., 34., 35., 36., 37., 38., 39.,
       40., 41., 42., 43., 44., 45., 46., 47., 48., 49., 50., 51., 52.,
       53., 54., 55., 56., 57., 58., 59., 60., 61., 62., 63., 64., 65.,
```

```
       66., 67., 68., 69., 70., 71., 72., 73., 74., 75., 76., 77., 78.,
       79., 80., 81., 82., 83., 84., 85., 86., 87., 88., 89., 90., 91.,
       92., 93., 94., 95., 96., 97., 98., 99.])
```

[241]:
```
np.arange(1,100,5)     # step size = 5
```

[241]:
```
array([ 1,  6, 11, 16, 21, 26, 31, 36, 41, 46, 51, 56, 61, 66, 71, 76, 81,
       86, 91, 96])
```

[242]:
```
np.arange(0,100,5)
```

[242]:
```
array([ 0,  5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80,
       85, 90, 95])
```

[243]:
```
np.linspace(1,10, 500)  # in a range 1 to 10 , it will return 500 random values
```

[243]:
```
array([ 1.        ,  1.01803607,  1.03607214,  1.05410822,  1.07214429,
        1.09018036,  1.10821643,  1.12625251,  1.14428858,  1.16232465,
        1.18036072,  1.19839679,  1.21643287,  1.23446894,  1.25250501,
        1.27054108,  1.28857715,  1.30661323,  1.3246493 ,  1.34268537,
        1.36072144,  1.37875752,  1.39679359,  1.41482966,  1.43286573,
        1.4509018 ,  1.46893788,  1.48697395,  1.50501002,  1.52304609,
        1.54108216,  1.55911824,  1.57715431,  1.59519038,  1.61322645,
        1.63126253,  1.6492986 ,  1.66733467,  1.68537074,  1.70340681,
        1.72144289,  1.73947896,  1.75751503,  1.7755511 ,  1.79358717,
        1.81162325,  1.82965932,  1.84769539,  1.86573146,  1.88376754,
        1.90180361,  1.91983968,  1.93787575,  1.95591182,  1.9739479 ,
        1.99198397,  2.01002004,  2.02805611,  2.04609218,  2.06412826,
        2.08216433,  2.1002004 ,  2.11823647,  2.13627255,  2.15430862,
        2.17234469,  2.19038076,  2.20841683,  2.22645291,  2.24448898,
        2.26252505,  2.28056112,  2.29859719,  2.31663327,  2.33466934,
        2.35270541,  2.37074148,  2.38877756,  2.40681363,  2.4248497 ,
        2.44288577,  2.46092184,  2.47895792,  2.49699399,  2.51503006,
        2.53306613,  2.5511022 ,  2.56913828,  2.58717435,  2.60521042,
        2.62324649,  2.64128257,  2.65931864,  2.67735471,  2.69539078,
        2.71342685,  2.73146293,  2.749499  ,  2.76753507,  2.78557114,
        2.80360721,  2.82164329,  2.83967936,  2.85771543,  2.8757515 ,
        2.89378758,  2.91182365,  2.92985972,  2.94789579,  2.96593186,
        2.98396794,  3.00200401,  3.02004008,  3.03807615,  3.05611222,
        3.0741483 ,  3.09218437,  3.11022044,  3.12825651,  3.14629259,
        3.16432866,  3.18236473,  3.2004008 ,  3.21843687,  3.23647295,
        3.25450902,  3.27254509,  3.29058116,  3.30861723,  3.32665331,
        3.34468938,  3.36272545,  3.38076152,  3.3987976 ,  3.41683367,
        3.43486974,  3.45290581,  3.47094188,  3.48897796,  3.50701403,
        3.5250501 ,  3.54308617,  3.56112224,  3.57915832,  3.59719439,
        3.61523046,  3.63326653,  3.65130261,  3.66933868,  3.68737475,
        3.70541082,  3.72344689,  3.74148297,  3.75951904,  3.77755511,
```

```
3.79559118,  3.81362725,  3.83166333,  3.8496994 ,  3.86773547,
3.88577154,  3.90380762,  3.92184369,  3.93987976,  3.95791583,
3.9759519 ,  3.99398798,  4.01202405,  4.03006012,  4.04809619,
4.06613226,  4.08416834,  4.10220441,  4.12024048,  4.13827655,
4.15631263,  4.1743487 ,  4.19238477,  4.21042084,  4.22845691,
4.24649299,  4.26452906,  4.28256513,  4.3006012 ,  4.31863727,
4.33667335,  4.35470942,  4.37274549,  4.39078156,  4.40881764,
4.42685371,  4.44488978,  4.46292585,  4.48096192,  4.498998  ,
4.51703407,  4.53507014,  4.55310621,  4.57114228,  4.58917836,
4.60721443,  4.6252505 ,  4.64328657,  4.66132265,  4.67935872,
4.69739479,  4.71543086,  4.73346693,  4.75150301,  4.76953908,
4.78757515,  4.80561122,  4.82364729,  4.84168337,  4.85971944,
4.87775551,  4.89579158,  4.91382766,  4.93186373,  4.9498998 ,
4.96793587,  4.98597194,  5.00400802,  5.02204409,  5.04008016,
5.05811623,  5.0761523 ,  5.09418838,  5.11222445,  5.13026052,
5.14829659,  5.16633267,  5.18436874,  5.20240481,  5.22044088,
5.23847695,  5.25651303,  5.2745491 ,  5.29258517,  5.31062124,
5.32865731,  5.34669339,  5.36472946,  5.38276553,  5.4008016 ,
5.41883768,  5.43687375,  5.45490982,  5.47294589,  5.49098196,
5.50901804,  5.52705411,  5.54509018,  5.56312625,  5.58116232,
5.5991984 ,  5.61723447,  5.63527054,  5.65330661,  5.67134269,
5.68937876,  5.70741483,  5.7254509 ,  5.74348697,  5.76152305,
5.77955912,  5.79759519,  5.81563126,  5.83366733,  5.85170341,
5.86973948,  5.88777555,  5.90581162,  5.9238477 ,  5.94188377,
5.95991984,  5.97795591,  5.99599198,  6.01402806,  6.03206413,
6.0501002 ,  6.06813627,  6.08617234,  6.10420842,  6.12224449,
6.14028056,  6.15831663,  6.17635271,  6.19438878,  6.21242485,
6.23046092,  6.24849699,  6.26653307,  6.28456914,  6.30260521,
6.32064128,  6.33867735,  6.35671343,  6.3747495 ,  6.39278557,
6.41082164,  6.42885772,  6.44689379,  6.46492986,  6.48296593,
6.501002  ,  6.51903808,  6.53707415,  6.55511022,  6.57314629,
6.59118236,  6.60921844,  6.62725451,  6.64529058,  6.66332665,
6.68136273,  6.6993988 ,  6.71743487,  6.73547094,  6.75350701,
6.77154309,  6.78957916,  6.80761523,  6.8256513 ,  6.84368737,
6.86172345,  6.87975952,  6.89779559,  6.91583166,  6.93386774,
6.95190381,  6.96993988,  6.98797595,  7.00601202,  7.0240481 ,
7.04208417,  7.06012024,  7.07815631,  7.09619238,  7.11422846,
7.13226453,  7.1503006 ,  7.16833667,  7.18637275,  7.20440882,
7.22244489,  7.24048096,  7.25851703,  7.27655311,  7.29458918,
7.31262525,  7.33066132,  7.34869739,  7.36673347,  7.38476954,
7.40280561,  7.42084168,  7.43887776,  7.45691383,  7.4749499 ,
7.49298597,  7.51102204,  7.52905812,  7.54709419,  7.56513026,
7.58316633,  7.6012024 ,  7.61923848,  7.63727455,  7.65531062,
7.67334669,  7.69138277,  7.70941884,  7.72745491,  7.74549098,
7.76352705,  7.78156313,  7.7995992 ,  7.81763527,  7.83567134,
7.85370741,  7.87174349,  7.88977956,  7.90781563,  7.9258517 ,
7.94388778,  7.96192385,  7.97995992,  7.99799599,  8.01603206,
```

```
        8.03406814,  8.05210421,  8.07014028,  8.08817635,  8.10621242,
        8.1242485 ,  8.14228457,  8.16032064,  8.17835671,  8.19639279,
        8.21442886,  8.23246493,  8.250501  ,  8.26853707,  8.28657315,
        8.30460922,  8.32264529,  8.34068136,  8.35871743,  8.37675351,
        8.39478958,  8.41282565,  8.43086172,  8.4488978 ,  8.46693387,
        8.48496994,  8.50300601,  8.52104208,  8.53907816,  8.55711423,
        8.5751503 ,  8.59318637,  8.61122244,  8.62925852,  8.64729459,
        8.66533066,  8.68336673,  8.70140281,  8.71943888,  8.73747495,
        8.75551102,  8.77354709,  8.79158317,  8.80961924,  8.82765531,
        8.84569138,  8.86372745,  8.88176353,  8.8997996 ,  8.91783567,
        8.93587174,  8.95390782,  8.97194389,  8.98997996,  9.00801603,
        9.0260521 ,  9.04408818,  9.06212425,  9.08016032,  9.09819639,
        9.11623246,  9.13426854,  9.15230461,  9.17034068,  9.18837675,
        9.20641283,  9.2244489 ,  9.24248497,  9.26052104,  9.27855711,
        9.29659319,  9.31462926,  9.33266533,  9.3507014 ,  9.36873747,
        9.38677355,  9.40480962,  9.42284569,  9.44088176,  9.45891784,
        9.47695391,  9.49498998,  9.51302605,  9.53106212,  9.5490982 ,
        9.56713427,  9.58517034,  9.60320641,  9.62124248,  9.63927856,
        9.65731463,  9.6753507 ,  9.69338677,  9.71142285,  9.72945892,
        9.74749499,  9.76553106,  9.78356713,  9.80160321,  9.81963928,
        9.83767535,  9.85571142,  9.87374749,  9.89178357,  9.90981964,
        9.92785571,  9.94589178,  9.96392786,  9.98196393, 10.        ])
```

arange return value within a range with or without step size, linspace return values in a range with asked number of values to be returned

[244]:
```
np.random.randint(2,4)
```

[244]: 2

[245]:
```
np.random.randint(2,10)
```

[245]: 9

[246]:
```
np.random.randn(50)   # As I want 50 random value
```

[246]:
```
array([ 1.77630421,  0.39695352,  0.85524111,  0.49470679,  2.30534329,
        1.85778956, -0.88051665, -0.86345696,  0.82219775, -1.06381997,
       -1.83837996, -0.40070681,  1.36212598,  0.03144252, -0.1533264 ,
       -0.84961782,  0.58545555,  1.38398763,  0.49177306,  0.35627133,
       -0.82626635,  0.82058867,  0.95588847,  0.62285106, -1.0851947 ,
        2.68057443, -0.54311531,  1.23095128,  0.09124117,  0.89211524,
        0.13829639,  1.06779953, -0.19578355,  0.42332392,  1.06314035,
       -0.79043849, -1.66444017,  1.09093346, -0.99327429, -0.88771843,
       -1.44171957, -0.27142989, -0.04253723,  1.31420643, -0.21731895,
       -1.18357951, -2.26470393, -0.2374272 , -1.12227708, -0.87210727])
```

[247]:
```
np.random.randn(1,15, 5)
```

```
[247]: array([[[ 1.13388515,  1.01058711, -0.71331156, -0.90785645,
          0.37841628],
        [-0.41054067,  0.78227455,  0.43189647,  0.69796182,
         -0.10715658],
        [-0.32953489, -3.34885829, -0.39564759, -0.32276598,
         -1.25813028],
        [-0.07000501,  1.10028008,  0.17893464, -1.08367028,
         -0.03232276],
        [ 0.14121705,  0.14594727,  0.52865128, -0.19762642,
          1.84609954],
        [ 0.76905834,  0.05845193,  1.20505759, -0.86817365,
         -0.21198211],
        [-0.9342976 , -1.06578026, -1.54075949, -0.18750926,
          0.70159695],
        [-2.77660279, -0.24751542, -0.54938195, -0.42044022,
         -0.34332651],
        [-0.81513893, -1.77752614, -0.40511996,  1.39549659,
          0.908286  ],
        [-0.05355661,  0.1661128 ,  0.36140393,  0.55195094,
          1.05594601],
        [ 0.41778725, -1.59272311, -1.91974306, -1.52853717,
         -0.47794275],
        [ 0.3010704 , -1.55143893, -0.06601224,  1.78681445,
          1.40836508],
        [-1.36582295, -1.84973942, -0.58023977,  1.25272668,
         -0.37848906],
        [-1.13349134,  0.58334225, -0.17400152, -0.64870417,
          0.78878933],
        [-0.40858952, -0.07942866,  0.42210569, -0.65646656,
          0.53936235]]])
```

4 Types of Array

- Convert any kind of iterable object into list, array mean numpy array
- by default it takes int types
- dtype can be mentioned as per need

```
[248]: interger = np.array([2,3,5,6,23,56,58])  # default integer
       interger
```

```
[248]: array([ 2,  3,  5,  6, 23, 56, 58])
```

```
[249]: defined_integer = np.array([2,3,4,5,5,50,75,80,80], dtype=np.int64)   # mention␣
        ↪dtype as int64
       defined_integer
```

```
[249]: array([ 2,  3,  4,  5,  5, 50, 75, 80, 80], dtype=int64)
```

[250]:
```
float = np.array([2,3,4,5,5,50,75,80,80], dtype=np.float64)  # mention dtype as␣
 ↪float64
float
```

[250]:
```
array([ 2.,  3.,  4.,  5.,  5., 50., 75., 80., 80.])
```

[251]:
```
bool = np.array([True, False, True, True, False], dtype=np.bool8)  # dtype as␣
 ↪boolean
bool
```

[251]:
```
array([ True, False,  True,  True, False])
```

[252]:
```
empty = np.empty([3,3], dtype=np.complex64)
empty
```

[252]:
```
array([[0.+2.j       , 0.+2.125j    , 0.+2.25j     ],
       [0.+2.3125j   , 0.+2.3125j   , 0.+3.140625j ],
       [0.+3.2929688j, 0.+3.3125j   , 0.+3.3125j   ]], dtype=complex64)
```

[253]:
```
string = np.array(['apple','orange', 'banana', 'grapes'], dtype=np.string_)  #␣
 ↪string type, mentioning dtype is recommended
string
```

[253]:
```
array([b'apple', b'orange', b'banana', b'grapes'], dtype='|S6')
```

5 NDarray Object Inspection

- ndim : there is function that is 'ndim' by which we can extract/ find dimention
- shape : by which we can know the shape (number of rows & columns)
- In NDarray object Inspection, type(ndim), shape is very important

[254]:
```
string.ndim
```

[254]:
```
1
```

[255]:
```
empty
```

[255]:
```
array([[0.+2.j       , 0.+2.125j    , 0.+2.25j     ],
       [0.+2.3125j   , 0.+2.3125j   , 0.+3.140625j ],
       [0.+3.2929688j, 0.+3.3125j   , 0.+3.3125j   ]], dtype=complex64)
```

[256]:
```
empty.ndim  # lets verify the the empty output, there a two bracket inside␣
 ↪array or array inside array, value as per rows & columns
```

[256]:
```
2
```

[257]:
```
empty.shape  # In order to verify the number of rows & columns
```

[257]: (3, 3)

[258]:
```
interger
```

[258]: array([2, 3, 5, 6, 23, 56, 58])

[259]:
```
interger.shape  # 7 value with 1 dimension
```

[259]: (7,)

[260]:
```
empty.itemsize # Represent the byte size of each item
string.itemsize
```

[260]: 6

[261]:
```
string.item(0) # checking index value, we can fetch data more easily
```

[261]: b'apple'

[262]:
```
bool
```

[262]: array([True, False, True, True, False])

[263]:
```
bool.any()  # Check & verify wheter any true in bool variable, its returning␣
↪true becasue of lots of true available in bool
```

[263]: True

[264]:
```
bool.all() # Check whether all true or not. .any & .all function basically␣
↪applied on boolean value
```

[264]: False

[265]:
```
string.T # Transpose its not properloy understandable because of 1D, it will␣
↪more clear when apply on 2D
```

[265]: array([b'apple', b'orange', b'banana', b'grapes'], dtype='|S6')

[266]:
```
# Checking Data type
empty.dtype
```

[266]: dtype('complex64')

[267]:
```
interger.dtype
```

[267]: dtype('int32')

[268]:
```
interger.itemsize
```

[268]: 4

[269]:
```
# We can directly inspect data type from here & depending on that we can decide
↪what kind of steps to be applied for this.becasue we cant run numerical
↪operation on str, so this will help to first understand
string.dtype
```

[269]: dtype('S6')

[270]:
```
empty.size # to know the total number of elements. empty is 3x3 matrix so total
↪9 elements
```

[270]: 9

- So, up to this we can slice & fetch, but there is another method by which we can do it more efficiently, that is more efficient than list slicing

6 Slicing

[271]:
```
arr = np.array(
    [
        [44, 22, 56],    # 1st Index
        [67, 76, 89],    # 2nd Index
        [52, 45, 78]     # 3rd Index
    ]
)
```

6.0.1 General Indexing

[272]:
```
arr[0]  # First Index Value
arr[1]  # 2nd index value
arr[2]  # 3rd index value
```

[272]: array([52, 45, 78])

[273]:
```
arr[0][0]  # First Index first value
arr[0][1]  # First Index 2nd Value
arr[0][2]  # First Index 3rd Value
```

[273]: 56

[274]:
```
arr[1][0]  # 2nd Index 1st value
arr[1][1]  # 2nd Index 2nd value
arr[1][2]  # 2nd Index 3rd value
```

[274]: 89

[275]:
```
arr[2][0] # 3rd index 1st Value
arr[2][1] # 3rd index 2nd Value
arr[2][2] # 3rd index 3rd Value
```

[275]: 78

6.0.2 Alternative Indexing

[276]:
```
arr
```

[276]:
```
array([[44, 22, 56],
       [67, 76, 89],
       [52, 45, 78]])
```

[277]:
```
arr[0][0] # General Indexing for 1st index 1st value
```

[277]: 44

[278]:
```
arr[0,0] # Alterative indexing for 1st index 1st value
```

[278]: 44

[279]:
```
arr[0][1] # 1st index, 2nd value
arr[0,1] # 1st index, 2nd value- Alternative
```

[279]: 22

[280]:
```
arr [0] [2] # 1st index, 3rd value
arr [0,2] # 1st index, 3rd value - Alternative
```

[280]: 56

[281]:
```
arr[1] [1] # 2nd index, 2nd value
arr [1, 1] # 2nd index, 2nd value- Alternative
```

[281]: 76

[282]:
```
arr [1] [0] # 2nd index, 1st Value
arr [1,0] # 2nd index, 1st value
```

[282]: 67

[283]:
```
arr # review
```

[283]:
```
array([[44, 22, 56],
       [67, 76, 89],
       [52, 45, 78]])
```

[284]:
```
arr[1] [2] # 2nd index, 3rd value
arr [1,2] # 2nd index, 3rd value- Alternative
```

[284]: 89

[285]:
```
arr [2] [0] # 3rd index, 1st value
arr [2,0]
```

[285]: 52

[286]:
```
arr[2][1] # 3rd index, 2nd value
arr[2,1] # 3rd index, 2nd value- Alternative
```

[286]: 45

[287]:
```
arr[2] [2] # 3rd index, 3rd value
arr [2,2] # 3rd index, 3rd value- Alternative
```

[287]: 78

[288]:
```
arr
```

[288]:
```
array([[44, 22, 56],
       [67, 76, 89],
       [52, 45, 78]])
```

[289]:
```
arr[0] # First Index Value/ 1st full row
arr[:,0] # First Index Value/ 1st full row - Alternative
```

[289]: array([44, 67, 52])

[290]:
```
arr[1] # 2nd Index Value/ full row
arr[:,1]
```

[290]: array([22, 76, 45])

[291]:
```
arr[1]
```

[291]: array([67, 76, 89])

[292]:
```
arr[2]
```

[292]: array([52, 45, 78])

[293]:
```
arr[:,0] # all rows: 1st column
```

[293]: array([44, 67, 52])

[294]:
```
arr[:,1] # all rows: 2nd column, comma means numpy
```

[294]: array([22, 76, 45])

[295]:
```
arr
```

[295]:
```
array([[44, 22, 56],
       [67, 76, 89],
       [52, 45, 78]])
```

[296]:
```
arr[:,2] # all rows: 3rd column
```

[296]: array([56, 89, 78])

[297]:
```
arr[:] # All
```

[297]:
```
array([[44, 22, 56],
       [67, 76, 89],
       [52, 45, 78]])
```

[298]:
```
arr[:,] # All Alternaive
```

[298]:
```
array([[44, 22, 56],
       [67, 76, 89],
       [52, 45, 78]])
```

6.0.3 Array 2 Practice

[299]:
```
arr2 = np.array(
    [
        [44, 22, 56, 71],    # 1st Index
        [67, 76, 89, 72],    # 2nd Index
        [52, 45, 78, 73],    # 3rd Index
    ]
)
```

[300]:
```
arr2[0,:]    # as per matrix, 0 is for row and : (colon) is for columns e.g all␣
↪columns
```

[300]: array([44, 22, 56, 71])

[301]:
```
arr2[1,:] # 2nd row, all columns
```

[301]: array([67, 76, 89, 72])

[302]:
```
arr2[2,:] # 3rd row/index, all columns
```

[302]: array([52, 45, 78, 73])

[303]:
```
arr2[0,:1] # firs row, first value
```

[303]: array([44])

[304]:
```
arr2[0,:2] # first row, first 2 values
```

[304]: array([44, 22])

[305]:
```
arr2[0,:3] # first row, first 3 values
```

[305]: array([44, 22, 56])

[306]:
```
arr2[0,:4] # first row, first 4 values
```

[306]: array([44, 22, 56, 71])

[307]:
```
arr2[0,:] # first row, first 4 values- Alternative
```

[307]: array([44, 22, 56, 71])

[308]:
```
arr2
```

[308]:
```
array([[44, 22, 56, 71],
       [67, 76, 89, 72],
       [52, 45, 78, 73]])
```

[309]:
```
arr2[1,:1] # 2nd row, 1st value
```

[309]: array([67])

[310]:
```
arr2[1,:2] # 2nd row, 1st 2 values
```

[310]: array([67, 76])

[311]:
```
arr2[1,:3] # 2nd row, first 3 values
```

[311]: array([67, 76, 89])

[312]:
```
arr2[1,:4] # 2nd row, fist 3 values
```

[312]: array([67, 76, 89, 72])

[313]:
```
arr2[1,:] # 2nd row, fist 3 values- Alternative
```

[313]: array([67, 76, 89, 72])

[314]:
```
arr2[2,:1] # 3rd row, 1 st value
arr2[2,:2] # 3rd row, first 2 st values
arr2[2,:3] # 3rd row, first 3 st values
arr2[2,:4] # 3rd row, first 4 st values
arr2[2,:]  # 3rd row, first 4 st values- Alternative
```

[314]: array([52, 45, 78, 73])

[315]:
```
arr2
```

[315]:
```
array([[44, 22, 56, 71],
       [67, 76, 89, 72],
       [52, 45, 78, 73]])
```

[316]:
```
arr2[0, 0:1] # 1st row, 1st index, 1st value
```

[316]: array([44])

[317]:
```
arr2[0,0:2] # 1st row, 1st to 2nd value
```

[317]: array([44, 22])

[318]:
```
arr2[0, 0:3] # 1st row, 1st to 3rd value
```

[318]: array([44, 22, 56])

[319]:
```
arr2 [0, 0:4] # # 1st row, 1st to 3rd value
```

[319]: array([44, 22, 56, 71])

[320]:
```
arr2 [2, 1:3]  # 3rd index/row, 2nd to 3rd value
```

[320]: array([45, 78])

[321]:
```
arr2[2, 2:4] # 3rd index/ row, 3rd to 4th value
```

[321]: array([78, 73])

[322]:
```
arr2
```

[322]:
```
array([[44, 22, 56, 71],
       [67, 76, 89, 72],
       [52, 45, 78, 73]])
```

[323]:
```
arr2[0, 1:3] # 1st index/row, 2nd to 3rd value
```

[323]: array([22, 56])

[324]:
```
arr2[1, 1:3] # 2nd index/ row,, 2nd to 3rd value
```

[324]: array([76, 89])

[325]:
```
arr2[1, 2:4]  # 2nd index/ row, 3rd to 4th value
```

[325]: array([89, 72])

[326]:
```
arr2[2, 2:4] # 3rd index/ value, 3rd to 4th value
```

[326]:
```
array([78, 73])
```

[327]:
```
arr2 [2, 1:4] # 3rd index/ value, 2nd to 4th value
```

[327]:
```
array([45, 78, 73])
```

[328]:
```
arr2[2, 1:3] # 3rd index/ value, 2nd to 3rd value
```

[328]:
```
array([45, 78])
```

6.0.4 Applying step size into indexcing

[329]:
```
arr2
```

[329]:
```
array([[44, 22, 56, 71],
       [67, 76, 89, 72],
       [52, 45, 78, 73]])
```

[330]:
```
arr2[0, 0:4:2]  # 1st row/ index, 1st to 4th value with 2 step in size  [0 to␣
↪4, 2 is step size]
```

[330]:
```
array([44, 56])
```

[331]:
```
arr2[0, 0:4:3] # 1st row/ index, 1st to 4th value with 3 step in size
```

[331]:
```
array([44, 71])
```

[332]:
```
arr2[1, 0:4:2] # 2nd row/ index, 1st to 4th value with 2 step in size
```

[332]:
```
array([67, 89])
```

[333]:
```
arr2[1, 0:4:3] # 2nd row/ index, 1st to 4th value with 3 step in size
```

[333]:
```
array([67, 72])
```

[334]:
```
arr2[2, 0:4:2] # 3rd row/ index, 1st to 4th value with 2 step in size
```

[334]:
```
array([52, 78])
```

[335]:
```
arr2[:]  # All rows : all columns
```

[335]:
```
array([[44, 22, 56, 71],
       [67, 76, 89, 72],
       [52, 45, 78, 73]])
```

[336]:
```
arr2[:, 0:4:2] # : means all rows, 0 to 4th with 2 in step size
```

```
[336]: array([[44, 56],
              [67, 89],
              [52, 78]])
```

```
[337]: arr2[:, 0:4:3]
```

```
[337]: array([[44, 71],
              [67, 72],
              [52, 73]])
```

```
[338]: arr2[:, 1:3] # Middle 2 rows & columns
```

```
[338]: array([[22, 56],
              [76, 89],
              [45, 78]])
```

```
[339]: arr2[1:, 0:4:2]  # 2nd index/ row 0 to 4th columns with 2 in step size
```

```
[339]: array([[67, 89],
              [52, 78]])
```

```
[340]: arr2[2:, 0:4:2] # 3rd index/ row , 0 to 4th columns with 2 in step size
```

```
[340]: array([[52, 78]])
```

```
[341]: arr2
```

```
[341]: array([[44, 22, 56, 71],
              [67, 76, 89, 72],
              [52, 45, 78, 73]])
```

```
[342]: arr2[1:, 0:4:2]
```

```
[342]: array([[67, 89],
              [52, 78]])
```

```
[343]: arr2[1:, 1:4:2] # 2nd index/ row; 1 to 4th columns, 2 in step size
```

```
[343]: array([[76, 72],
              [45, 73]])
```

```
[344]: arr2[0:, 1:4:2] # 1st index/ row, 1 to 4th columns with 2 in step size
```

```
[344]: array([[22, 71],
              [76, 72],
              [45, 73]])
```

```
[345]: arr2[1:, 1:3] # 2nd index/ row, 1 to 3rd colum, In such way using comma, we can
        ↪fetch multi dimentional arrays
```

[345]: array([[76, 89],
        [45, 78]])

```
[346]: arr2
```

[346]: array([[44, 22, 56, 71],
        [67, 76, 89, 72],
        [52, 45, 78, 73]])

```
[347]: arr2[1:3:2, 0:4:2] # 2nd row/ index: 1 to 3 columns 2 in step size,  1 to 4␣
    ↪colums with step 2 in size
```

[347]: array([[67, 89]])

```
[348]: arr2[1:, 2:4] # 2nd row/ index; 3 to 4 th columns
```

[348]: array([[89, 72],
        [78, 73]])

```
[349]: arr2[1:, 2:] # 2nd row/ index; 3 to 4 th columns- Alternative
```

[349]: array([[89, 72],
        [78, 73]])

```
[350]: arr3 = np.array(
           [
               [44, 22, 45, 33],
               [34, 45, 40, 32],
               [12, 56, 80, 90],
               [50, 40, 60, 70]
           ]
       )
```

```
[351]: arr3[1:3:, 1:3] # Middle 4 value, 2nd & 3rd row:, 2nd to 3rd colums.  1:3␣
    ↪Dimenstion One, 1:3 Dimenstion Two
```

[351]: array([[45, 40],
        [56, 80]])

```
[352]: arr3[1:3:, 1:]   # 2nd to 3rd rows:, 2 to last all columns
```

[352]: array([[45, 40, 32],
        [56, 80, 90]])

```
[353]: arr3[2:, 1:]  # from 3rd all rows, 2nd to last all columns
```

[353]: array([[56, 80, 90],
        [40, 60, 70]])

[357]:
```
arr3[2:, 1:3] # from 3rd all rows 2 to 3rd colums
```

[357]:
```
array([[56, 80],
       [40, 60]])
```

[355]:
```
arr3[1:, 2:]
```

[355]:
```
array([[40, 32],
       [80, 90],
       [60, 70]])
```

[356]:
```
arr3 [1:2:, 1:2]
```

[356]:
```
array([[45]])
```