```
In [ ]: import numpy as np

In [ ]: # integer of size 8 bytes
        myarr = np.array([[1,2, 3,4]], np.int64)

In [ ]: myarr.shape

Out[ ]: (1, 4)

In [ ]: myarr[0].shape

Out[ ]: (4,)

In [ ]: myarr.dtype

Out[ ]: dtype('int64')

In [ ]: myarr[0, 1]

Out[ ]: 2

In [ ]: myarr[0, 1] = 34

In [ ]: myarr

Out[ ]: array([[ 1, 34,  3,  4]])

In [ ]: # Array creation: Conversion from other Python structures (i.e. lists and

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

In [ ]: listarray.dtype

Out[ ]: dtype('int64')

In [ ]: listarray.shape

Out[ ]: (3, 3)

In [ ]: listarray.size

Out[ ]: 9

In [ ]: # Not recommended for calculation
        np.array({34,23,36})

Out[ ]: array({34, 36, 23}, dtype=object)

In [ ]: # array creation: Intrinsic NumPy array creation functions (e.g. arange,

In [ ]: zeros = np.zeros((2,5))

In [ ]: zeros
```

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

```
In [ ]:  zeros.shape
```

```
Out[ ]:  (2, 5)
```

```
In [ ]:  rng = np.arange(15)
```

```
In [ ]:  rng
```

```
Out[ ]:  array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14])
```

```
In [ ]:  lspace = np.linspace(1, 50, 12)
```

```
In [ ]:  lspace
```

```
Out[ ]:  array([ 1.        ,  5.45454545,  9.90909091, 14.36363636, 18.81818182,
                23.27272727, 27.72727273, 32.18181818, 36.63636364, 41.09090909,
                45.54545455, 50.        ])
```

```
In [ ]:  lspace = np.linspace(1, 4, 4)
         lspace
         lspace.dtype
```

```
Out[ ]:  dtype('float64')
```

```
In [ ]:  # Filled with random values. size 4,6
         emp = np.empty((4,6))
         emp
```

```
Out[ ]:  array([[2.27958174e-316, 0.00000000e+000, 9.82157975e+252,
                 8.89489936e+252, 6.01346954e-154, 6.01347002e-154],
                [6.01347002e-154, 6.01347002e-154, 9.08366793e+223,
                 1.14177168e+243, 2.45126797e+198, 1.06083187e-153],
                [2.35625393e+251, 6.01334511e-154, 6.01347002e-154,
                 6.01347002e-154, 6.01347002e-154, 1.88556770e+122],
                [4.96820036e+180, 6.80600993e+212, 1.10317376e+217,
                 1.19490107e+190, 2.06642651e+161, 5.44760669e-109]])
```

```
In [ ]:  # empty_like: Return a new array with the same shape and type as a given
         emp_like = np.empty_like(lspace)
         emp_like
```

```
Out[ ]:  array([1., 2., 3., 4.])
```

In [ ]:

```
# identity matrix
# creates a square matrix with ones on the diagonal and zeros elsewhere
# shape nxn
ide = np.identity(45)
ide
```

Out[ ]:

```
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.]])
```

In [ ]:

```
# reshape
arr = np.arange(99)

arr = arr.reshape(3,33)
arr
```

Out[ ]:

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

In [ ]:

```
# again making it 1D
arr.ravel()
```

Out[ ]:

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 1
6,
       17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 3
3,
       34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 5
0,
       51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 6
7,
       68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 8
4,
       85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98])
```

In [ ]:

```
# numpy axis
# axis 0 is vertical axis 1 is horizontal
# For 2D array, axis 0 is vertical (along the rows) and axis 1 is horizon
# For 1D array, axis 0 is horizontal
```

In [ ]:

```
x = [[1,2,3],
     [4,5,6],
     [7,1,0]]
arr = np.array(x)
arr.sum(axis=0)
```

Out[ ]:

```
array([12,  8,  9])
```

In [ ]:

```
arr.sum(axis=1)
```

```
Out[ ]:  array([ 6, 15,  8])
```

```
In [ ]: # For transpose
        arr.T
```

```
Out[ ]:  array([[1, 4, 7],
                [2, 5, 1],
                [3, 6, 0]])
```

```
In [ ]: # For flat. It returns a 1D iterator over the array
        for item in arr.flat:
            print(item)
```

```
1
2
3
4
5
6
7
1
0
```

```
In [ ]: # ndim - number of dimensions
        arr.ndim
```

```
Out[ ]:  2
```

```
In [ ]: arr.size
```

```
Out[ ]:  9
```

```
In [ ]: arr.nbytes
```

```
Out[ ]:  72
```

```
In [ ]: one = np.array([1, 3, 4, 634, 23])
```

```
In [ ]: # argmax - returns the index of the maximum value in the array
        one.argmax()
```

```
Out[ ]:  3
```

```
In [ ]: # argmin - returns the index of the minimum value in the array
        one.argmin()
```

```
Out[ ]:  0
```

```
In [ ]: # argsort - returns the indices that would sort the array
        one.argsort()
```

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

```
In [ ]: # for 2d array
        arr
```

```
Out[ ]:  array([[1, 2, 3],
                [4, 5, 6],
                [7, 1, 0]])

In [ ]:  arr.argmin()

Out[ ]:  8

In [ ]:  arr.argmax()

Out[ ]:  6

In [ ]:  arr.argmax(axis=0)

Out[ ]:  array([2, 1, 1])

In [ ]:  arr.argmax(axis=1)

Out[ ]:  array([2, 2, 0])

In [ ]:  arr.argsort(axis=1)

Out[ ]:  array([[0, 1, 2],
                [0, 1, 2],
                [2, 1, 0]])

In [ ]:  arr.argsort(axis=0)

Out[ ]:  array([[0, 2, 2],
                [1, 0, 0],
                [2, 1, 1]])

In [ ]:  arr.ravel()

Out[ ]:  array([1, 2, 3, 4, 5, 6, 7, 1, 0])

In [ ]:  arr.reshape(9,1)

Out[ ]:  array([[1],
                [2],
                [3],
                [4],
                [5],
                [6],
                [7],
                [1],
                [0]])

In [ ]:  arr

Out[ ]:  array([[1, 2, 3],
                [4, 5, 6],
                [7, 1, 0]])

In [ ]:  arr2 = np.array([[1,2,1],
                           [4,0,6],
                           [8,1,0]])
```

```
In [ ]: arr + arr2

Out[ ]: array([[ 2,  4,  4],
               [ 8,  5, 12],
               [15,  2,  0]])
```

```
In [ ]: [324, 23, 23] + [23, 23, 23]

Out[ ]: [324, 23, 23, 23, 23, 23]
```

```
In [ ]: # element wise multiplication
        arr * arr2

Out[ ]: array([[ 1,  4,  3],
               [16,  0, 36],
               [56,  1,  0]])
```

```
In [ ]: # Element wise square root
        np.sqrt(arr)

Out[ ]: array([[1.        , 1.41421356, 1.73205081],
               [2.        , 2.23606798, 2.44948974],
               [2.64575131, 1.        , 0.        ]])
```

```
In [ ]: arr.sum()

Out[ ]: 29
```

```
In [ ]: arr.max()

Out[ ]: 7
```

```
In [ ]: arr.min()

Out[ ]: 0
```

```
In [ ]: arr

Out[ ]: array([[1, 2, 3],
               [4, 5, 6],
               [7, 1, 0]])
```

```
In [ ]: # Position of max element. Returns tuple of indices
        np.where(arr>5)

Out[ ]: (array([1, 2]), array([2, 0]))
```

```
In [ ]: # returns the number of non-zero elements in the array
        np.count_nonzero(arr)

Out[ ]: 8
```

```
In [ ]: # returns tuples of indices of non-zero elements
        np.nonzero(arr)

Out[ ]: (array([0, 0, 0, 1, 1, 1, 2, 2]), array([0, 1, 2, 0, 1, 2, 0, 1]))
```

In [ ]:

```
# to list
arr.tolist()
```

Out[ ]:

```
[[1, 2, 3], [4, 5, 6], [7, 1, 0]]
```

In [ ]: