Numpy

Why use Numpy?

- Numpy provides efficient storage.
- It also provides better ways of handling data for processing.
- It is fast.
- It is easy to learn.
- Numpy uses relatively less memory to store data than the python objects.

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

In [2]:
```
myarr = np.array([1, 2, 3, 4, 5]) # Create a numpy array from list
```

In [3]:
```
myarr # View the array
```

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

In [4]:
```
# Memory management
myarr = np.array([1, 2, 3, 4, 5], np.int8) # Create a numpy array from list with in
```

In [5]:
```
# Accessing elements
myarr[0] # Access the first element of the array
```

Out[5]:
```
np.int8(1)
```

In [6]:
```
# Slicing
myarr[0:2] # Access the first 5 elements of the array
```

Out[6]:
```
array([1, 2], dtype=int8)
```

In [7]:
```
# Shape of the array
myarr.shape # Get the shape of the array
```

Out[7]:
```
(5,)
```

In [8]:
```
# 2D array
myarr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]) # Create a 2D array
```

In [9]:
```
# Accessing elements
myarr[0, 0] # Access the first element of the array
```

Out[9]:
```
np.int64(1)
```

In [10]:
```
# Slicing
myarr[0:2, 0:2] # Access the first
```

Out[10]:

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

In [11]:

```
myarr.shape # Get the shape of the array
```

Out[11]:

```
(3, 3)
```

In [12]:

```
# Changing the element
myarr[0, 0] = 99 # Change the first element of the array
```

Methods Of Creation

There are 5 general ways of creating a numpy array:

1. Conversion from other Python structures (e.g. lists, tuples)
2. Intrinsic numpy array creation objects (e.g. arange, ones, zeros,etc.)
3. Reading arrays from disk, either from standard or custom formats
4. Creating arrays from raw bytes through the use of strings or buffers
5. Use of special library functions (e.g. random)

1. Conversion from other Python structures (e.g., lists, tuples)

In [13]:

```
mylist = [1, 2, 3, 4, 5] # Create a list
myarr = np.array(mylist) # Convert the list to numpy array
myarr # View the array
```

Out[13]:

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

In [14]:

```
mydict = {1: 'a', 2: 'b', 3: 'c'} # Create a dictionary
np.array(mydict) # Convert the dictionary to numpy array
```

Out[14]:

```
array({1: 'a', 2: 'b', 3: 'c'}, dtype=object)
```

2. Intrinsic numpy array creation objects (e.g., arange, ones, zeros, etc.)

In [15]:

```
np.arange(0, 10, 2) # Create an array with elements from 0 to 10 with step 2
```

Out[15]:

```
array([0, 2, 4, 6, 8])
```

In [16]:

```
zeros = np.zeros((2, 3)) # Create a 2x3 array with zeros
ones = np.ones((2, 3)) # Create a 2x3 array with ones
zeros, ones # View the arrays
```

Out[16]:

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

In [17]:

```
lspace = np.linspace(1, 5, 10) # Create an array with 10 elements from 1 to 5 equal
lspace # View the array
```

Out[17]:

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

In [18]:

```
emp = np.empty((2, 3)) # Create an empty array with 2x3 shape
emp # View the array
```

Out[18]:

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

In [19]:

```
emp_like = np.empty_like(lspace) # Create an empty array with the same shape as lsp
emp_like # View the array
```

Out[19]:

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

In [20]:

```
ide = np.identity(45) # Create an identity matrix of 45x45
ide # View the matrix
```

Out[20]:

```
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.]], shape=(45, 45))
```

Attributes

In [21]:

```
# Find the shape of the array
ide.shape # Get the shape of the matrix
```

Out[21]:

```
(45, 45)
```

In [22]:

```
# Find the dimension of the array
ide.ndim # Get the dimension of the matrix
```

Out[22]:

```
2
```

In [23]:

```
# Find the data type of the array
ide.dtype # Get the data type of the matrix
```

Out[23]:

```
dtype('float64')
```

In [24]:

```
# Total bytes consumed by an array
myarr.nbytes # Get the total bytes consumed by the array
```

Out[24]:

```
40
```

In [25]:

```
# Transpose the matrix
myarr = np.array([[1, 2], [3, 4]])
```

```
myarr.T
```

Out[25]:

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

In [26]:

```
myarr.flat # Get the iterator of the array
```

Out[26]:

```
<numpy.flatiter at 0x1a7751e1360>
```

In [27]:

```
for item in myarr.flat: # Iterate through the array
    print(item) # Print the item
```

```
1
2
3
4
```

Methods

reshape(x, y, ...)

Reshapes the given array. It doesn't add or remove any element.

In [28]:

```
arr = np.arange(99) # Create an array with elements from 0 to 99
arr # View the array
```

Out[28]:

```
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 [29]:

```
arr.reshape(3, 33) # Reshape the array to 3x33
```

Out[29]:

```
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]])
```

ravel()

Converts an array into 1D array.

In [30]:

```
arr = np.arange(99).reshape(3, 33) # Create an array with elements from 0 to 99 and
arr # View the array
```

```
Out[30]: 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 [31]: arr.ravel() # Flatten the array
```

```
Out[31]: 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 [32]: # Maximum element index in the array
         arr.argmax() # Get the maximum element
```

```
Out[32]: np.int64(98)
```

```
In [33]: # Minimum element index in the array
         arr.argmin() # Get the minimum element
```

```
Out[33]: np.int64(0)
```

```
In [34]: # Sorted index of the arrray
         arr.argsort() # Get the sorted index of the array
```

```
Out[34]: 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],
                [ 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],
                [ 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]])
```

```
In [35]: np.where(arr > 5) # Get the index of elements greater than 50
```

```
Out[35]: (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, 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, 2, 2, 2, 2, 2, 2,
                 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
                 2, 2, 2, 2, 2]),
          array([ 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,  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,  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]))
```

```
In [36]: np.nonzero(arr) # Get the index of non-zero elements
```

```
Out[36]: (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, 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, 2,
                 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
                 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]),
          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,  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,  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]))
```

```
In [37]: np.count_nonzero(arr) # Count the number of non-zero elements in the array
```

```
Out[37]: 98
```

Axes

```
In [38]: x = [[1, 2, 3], [4, 5, 6], [7, 8, 9]] # Create a list
         arr = np.array(x) # Convert the list to numpy array
         arr # View the array
```

```
Out[38]: array([[1, 2, 3],
                [4, 5, 6],
                [7, 8, 9]])
```

```
In [39]: arr.sum(axis=0) # Sum of the elements along the column
```

```
Out[39]: array([12, 15, 18])
```

```
In [40]: arr.sum(axis=1) # Sum of the elements along the row
```

```
Out[40]: array([ 6, 15, 24])
```

```
In [41]: # Maximum element index in the axes
         arr.argmax(axis=1) # Get the maximum element
```

```
Out[41]: array([2, 2, 2])
```

```
In [42]: # Minimum element index in the axes
         arr.argmin(axis=0) # Get the minimum element
```

```
Out[42]: array([0, 0, 0])
```

```
In [43]: # Sorted index of the axes
         arr.argsort(axis=0) # Get the sorted index of the array
```

```
Out[43]: array([[0, 0, 0],
                [1, 1, 1],
                [2, 2, 2]])
```

Mathematical Operations

In [44]:

```
arr1 = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]) # Create a 2D array
arr2 = np.array([[9, 8, 7], [6, 5, 4], [3, 2, 1]]) # Create a 2D array
```

In [45]:

```
arr1 + arr2 # Add two arrays
```

Out[45]:

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

In [46]:

```
arr1 * arr2 # Multiply two arrays element-wise
```

Out[46]:

```
array([[ 9, 16, 21],
       [24, 25, 24],
       [21, 16,  9]])
```

In [47]:

```
np.sqrt(arr1) # Square root of the array
```

Out[47]:

```
array([[1.        , 1.41421356, 1.73205081],
       [2.        , 2.23606798, 2.44948974],
       [2.64575131, 2.82842712, 3.        ]])
```

In [48]:

```
arr1.sum() # Sum of the array
```

Out[48]:

```
np.int64(45)
```

In [49]:

```
arr1.max() # Maximum element of the array
```

Out[49]:

```
np.int64(9)
```

In [50]:

```
arr.min() # Minimum element of the array
```

Out[50]:

```
np.int64(1)
```

Official Documentation