Information and Communication Technologies in Transport

NumPy (Numerical Python)

Engr. Saeed Ahmad

What is Numpy?

- NumPy is a free and open source Python library that will help you manipulate numbers and apply numerical functions
- They are similar to python lists

How to Install and Import?

The numpy library can be installed in V.S code as follows:

- pip install Numpy

The numpy library can be import in V.S code as follows:

- Import numpy as np

Difference Between Numpy and Python List

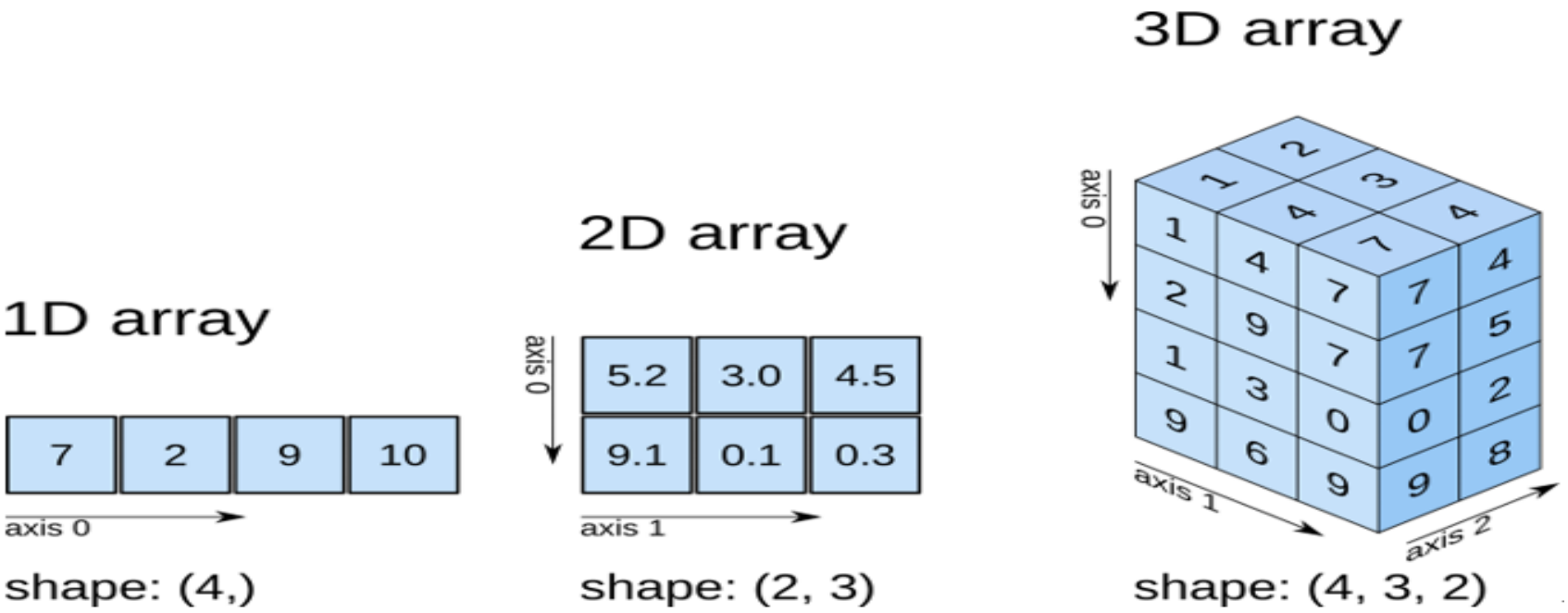
Numpy	Python Lists
Same Data type	Different Data type
Great for big data operations	Best for short codes
Consume less memory and it is convenient to use	Its much less efficient

What is Array in Numpy?

- An array is a central data structures of numpy library
- Array is a grid of values and it contains information about the raw data, how to locate an element and how to interpret an element.
- It has grid of elements that can be indexed in various ways
- The elements are all of same type, referred as array dtype.
- Array can be indexed by tuple of non-negative integers by Booleans by another array or by integers

Rank of Array

- The rank of array is the number of axes or dimensions it has. A simple list/array has rank 1 (called a Vector)
- 2 dimensional array (sometimes called a matrix) has rank 2
- 3 dimensional array (sometimes called a Tensor) has rank 3.



Shape of Array

- The shape of an array is the number of elements in each dimension
- It returns a tuple with each index having the number of corresponding elements.

```
1  a = np.array([1,2,3])
2  a.shape
```
✓ 0.1s

```
(3,)
```

```
1  b = np.array([[1,2,3],[4,5,6]])
2  b
```
✓ 0.0s

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

```
1  b = np.array([[1,2,3],[4,5,6]])
2  b.shape
```
✓ 0.0s

```
(2, 3)
```

How to make 1 D Array?

1-D array:

```
In [9]:
1 import numpy as np
2 a = np.array([5,5,5,])
3 a

Out[9]: array([5, 5, 5])

In [6]:
1 type(a)

Out[6]: numpy.ndarray

In [16]:
1 len(a)

Out[16]: 3

In [13]:
1 a[0]

Out[13]: 5

In [14]:
1 a[0:]

Out[14]: array([5, 5, 5])
```

How to make 1 D Array?

Initial Placeholders

`>>> np.zeros((3,4))`	Create an array of zeros
`>>> np.ones((2,3,4),dtype=np.int16)`	Create an array of ones
`>>> d = np.arange(10,25,5)`	Create an array of evenly spaced values (step value)
`>>> np.linspace(0,2,9)`	Create an array of evenly spaced values (number of samples)
`>>> e = np.full((2,2),7)`	Create a constant array
`>>> f = np.eye(2)`	Create a 2X2 identity matrix
`>>> np.random.random((2,2))`	Create an array with random values
`>>> np.empty((3,2))`	Create an empty array

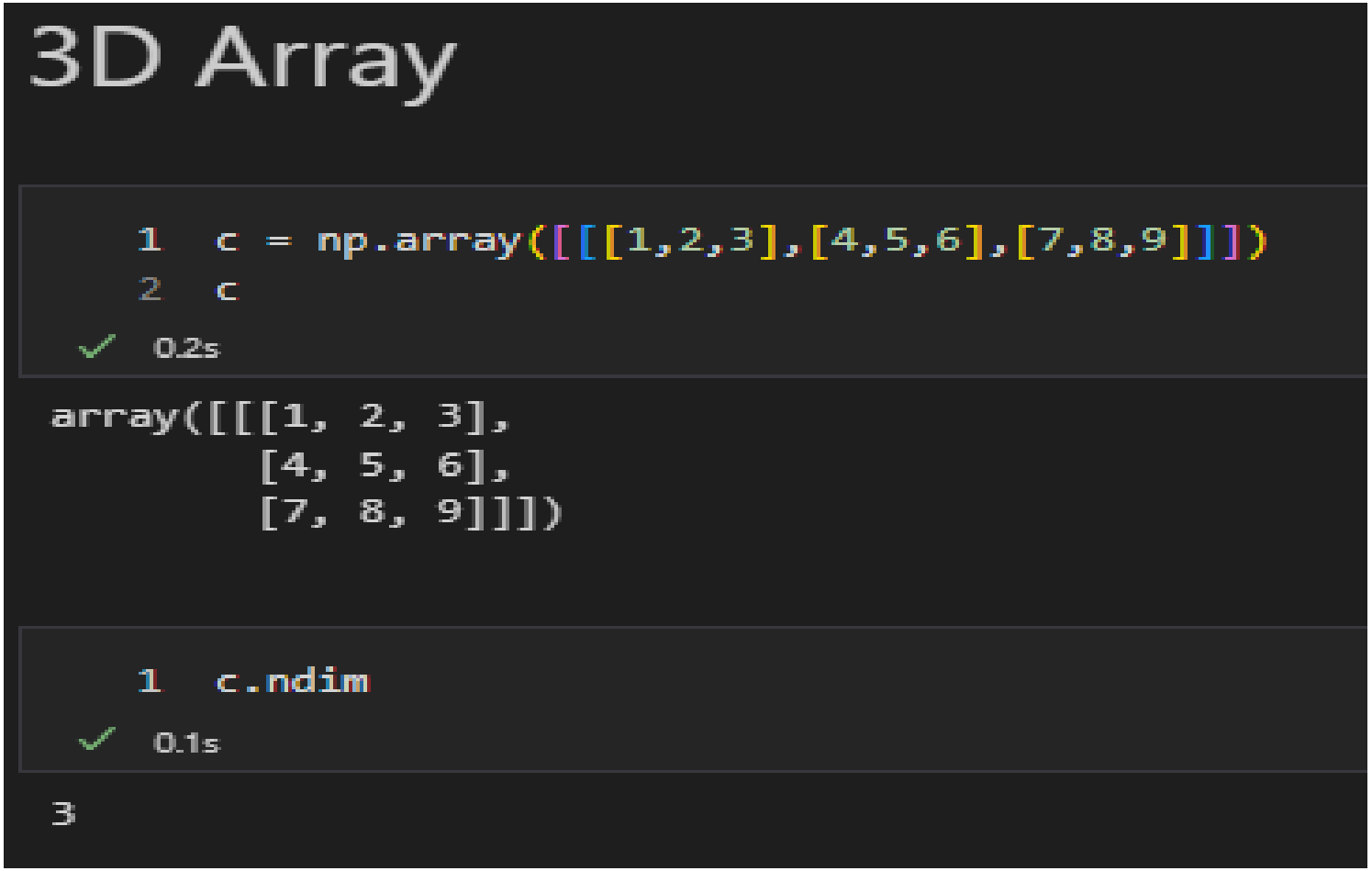
How to make 2 D Array?

2-D array:

```
In [18]:  1 # list of lists
          2 b = np.array([[5,5,5], [5, 5,5], [5,5,5]])
          3 b

Out[18]: array([[5, 5, 5],
                [5, 5, 5],
                [5, 5, 5]])

In [19]:  1 type(b)

Out[19]: numpy.ndarray

In [20]:  1 len(b)

Out[20]: 3

In [21]:  1 b[0]

Out[21]: array([5, 5, 5])

In [22]:  1 b[0:]

Out[22]: array([[5, 5, 5],
                [5, 5, 5],
                [5, 5, 5]])
```

A matrix refers to an array with two dimensions.

How to make 3 D Array?



Array Functions

- 1- Arithmetic Operators
- 2- Comparison
- 3- Aggregate Functions
- 4- Indexing
- 5- Slicing

NumPy

- Stands for Numerical Python
- Is the fundamental package required for high performance computing and data analysis
- NumPy is so important for numerical computations in Python is because it is designed for efficiency on large arrays of data.
- It provides
  - ndarray for creating multiple dimensional arrays
  - Internally stores data in a contiguous block of memory, independent of other built-in Python objects, use much less memory than built-in Python sequences.
  - Standard math functions for fast operations on entire arrays of data without having to write loops
  - NumPy Arrays are important because they enable you to express batch operations on data without writing any *for* loops. We call this *vectorization*.

NumPy ndarray vs list

- One of the key features of NumPy is its N-dimensional array object, or ndarray, which is a fast, flexible container for large datasets in Python.
- Whenever you see “array,” “NumPy array,” or “ndarray” in the text, with few exceptions they all refer to the same thing: the ndarray object.
- NumPy-based algorithms are generally 10 to 100 times faster (or more) than their pure Python counterparts and use significantly less memory.

```
import numpy as np
my_arr = np.arange(1000000)
my_list = list(range(1000000))
```

ndarray

- ndarray is used for storage of homogeneous data
  - i.e., all elements the same type
- Every array must have a shape and a dtype
- Supports convenient slicing, indexing and efficient vectorized computation

```
import numpy as np
data1 = [6, 7.5, 8, 0, 1]
arr1 =  np.array(data1)
print(arr1)
print(arr1.dtype)
print(arr1.shape)
print(arr1.ndim)
```

Creating ndarrays

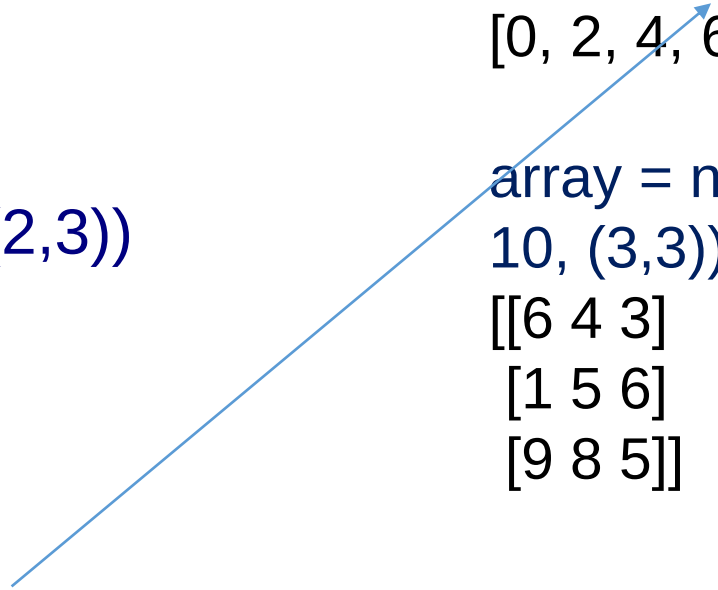
Using list of lists

```
import numpy as np

data2 = [[1, 2, 3, 4], [5, 6, 7, 8]]  #list of lists
arr2 = np.array(data2)
print(arr2.ndim)   #2
print(arr2.shape)  # (2,4)
```

Creating ndarrays

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

array = np.zeros((2,3))
[[0. 0. 0.]
 [0. 0. 0.]]

array = np.ones((2,3))
[[1. 1. 1.]
 [1. 1. 1.]]
```

```
array = np.eye(3)
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]

array = np.arange(0, 10, 2)
[0, 2, 4, 6, 8]

array = np.random.randint(0,
10, (3,3))
[[6 4 3]
 [1 5 6]
 [9 8 5]]
```



arange is an array-valued version of the built-in Python range function

Arithmatic with NumPy Arrays

- Any arithmetic operations between equal-size arrays applies the operation element-wise:

```
arr = np.array([[1., 2., 3.], [4., 5., 6.]])
print(arr)
[[1. 2. 3.]
 [4. 5. 6.]]
print(arr * arr)
[[ 1.  4.  9.]
 [16. 25. 36.]]
print(arr - arr)
[[0. 0. 0.]
 [0. 0. 0.]]
```

Arithmatic with NumPy Arrays

- Arithmetic operations with scalars propagate the scalar argument to each element in the array:

```
arr = np.array([[1., 2., 3.], [4., 5., 6.]])
print(arr)
[[1. 2. 3.]
 [4. 5. 6.]]
print(arr **2)
[[ 1.  4.  9.]
 [16. 25. 36.]]
```

- Comparisons between arrays of the same size yield boolean arrays:

```
arr2 = np.array([[0., 4., 1.], [7., 2., 12.]])
print(arr2)
[[ 0.  4.  1.]
 [ 7.  2. 12.]]

print(arr2 > arr)
[[False  True False]
 [ True False  True]]
```

Indexing and Slicing

- One-dimensional arrays are simple; on the surface they act similarly to Python lists:

```
arr = np.arange(10)
print(arr)          # [0 1 2 3 4 5 6 7 8 9]
print(arr[5])      #5
print(arr[5:8]) #[5 6 7]
arr[5:8] = 12
print(arr)      #[ 0 1 2 3 4 12 12 12 8 9]
```

Indexing and Slicing

- As you can see, if you assign a scalar value to a slice, as in arr[5:8] = 12, the value is propagated (or *broadcasted*) to the entire selection.
- An important first distinction from Python's built-in lists is that array slices are *views* on the original array.
  - This means that the data is not copied, and any modifications to the view will be reflected in the source array.

```
arr = np.arange(10)
print(arr)          # [0 1 2 3 4 5 6 7 8 9]

arr_slice = arr[5:8]
print(arr_slice)            # [5 6 7]
arr_slice[1] = 12345
print(arr)                  # [    0     1     2     3     4     5 12345     7     8     9]
arr_slice[:] = 64
print(arr)                  # [ 0  1  2  3  4 64 64 64  8  9]
```

Indexing

- In a two-dimensional array, the elements at each index are no longer scalars but rather one-dimensional arrays:

```
arr2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(arr2d[2])        # [7 8 9]
```

- Thus, individual elements can be accessed recursively. But that is a bit too much work, so you can pass a comma-separated list of indices to select individual elements.
- So these are equivalent:

```
print(arr2d[0][2])      # 3
print(arr2d[0, 2])      #3
```

Activity 3

- Consider the two-dimensional array, arr2d.

```
arr2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
```

- Write a code to slice this array to display the last column,

  [[3] [6] [9]]

- Write a code to slice this array to display the last 2 elements of middle array,

  [5 6]