

## Week -11

**NumPy Brief about NumPy module; NumPy arithmetic functions; NumPy array manipulation functions; NumPy statistical functions; Pandas Introduction, series, data frame; Create dataframes; formatting data; fundamental data frame operations;**

### NumPy:

- ☐ NumPy is a Python library used for working with arrays.
- ☐ It also has functions for working in domain of linear algebra, Fourier transform, and matrices.
- ☐ NumPy was created in 2005 by Travis Oliphant.
- ☐ It is open-source and can be used freely.
- ☐ NumPy stands for Numerical Python.

### Why Use NumPy:

- ☐ In Python we have lists that serve the purpose of arrays, but they are slow to process.
- ☐ NumPy aims to provide an array object that is up to 50x faster than traditional Python lists.
- ☐ The array object in NumPy is called ndarray.
- ☐ It provides a lot of supporting functions that make working with ndarray very easy.
- ☐ Arrays are very frequently used in data science, where speed and resources are very important.

### Installing NumPy:

- ☐ NumPy can be installed using the python package manger pip via command-line.
- ☐ The command for installing NumPy is:

**pip install numpy**

### Numpy Arrays:

We can create a NumPy ndarray object by using the array() function.

Ex:

```
import numpy as np
arr = np.array([10, 23, 34, 46, 58])
print(arr)
print(type(arr))
```

```
output:
[10 23 34 46 58]
<class 'numpy.ndarray'>
```

## Dimensions in Arrays

A dimension in arrays is one level of array depth (nested arrays).

### 0-D Arrays

0-D arrays, or Scalars, are the elements in an array. Each value in an array is a 0-D array.

Example

Create a 0-D array with value 42

```
import numpy as np
arr = np.array(42)
print(arr)
```

42

### 1-D Arrays

An array that has 0-D arrays as its elements is called uni-dimensional or 1-D array.

These are the most common and basic arrays.

Example

Create a 1-D array containing the values 1,2,3,4,5:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
print(arr)
```

[1,2,3,4,5]

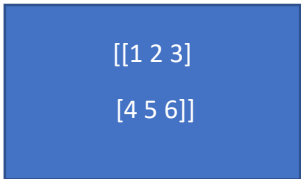
### 2-D Arrays

An array that has 1-D arrays as its elements is called a 2-D array.

Example

Create a 2-D array containing two arrays with the values 1,2,3 and 4,5,6:

```
import numpy as np
arr = np.array([[1, 2, 3], [4, 5, 6]])
print(arr)
```

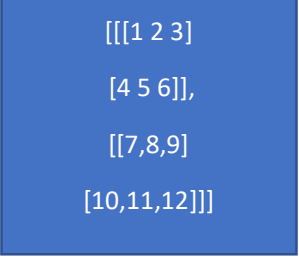
[[1 2 3]  
[4 5 6]]

### 3-D Arrays

An array that has 2-D arrays as its elements is called a 3-D array.

Example

```
import numpy as np
arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
print(arr)
```



```
[[[1 2 3]
  [4 5 6]],
 [[7, 8, 9]
  [10, 11, 12]]]
```

### Attributes of Numpy array

The contents of ndarray can be accessed and modified by indexing or slicing the array and via the methods and attributes of the ndarray. The more important attributes of ndarray object are shown in the following table:

ndarray Attributes	Description
ndarray.ndim	Gives the number of axes or dimensions in the array
ndarray.shape	Gives the dimensions of the array. For an array with n rows and m columns, shape will be a tuple of integers (n, m).
ndarray.size	Gives the total number of elements of the array.
ndarray.dtype	Gives an object describing the type of the elements in the array. One can create or specify dtype's using standard Python types. Additionally, NumPy provides its own types like np.int32, np.int16, np.float64, and others.
ndarray.itemsize	Gives the size of each element of the array in bytes.

### Access 1-D Array Elements

- Array indexing is the same as accessing an array element. You can access an array element by referring to its index number.
- The indexes in NumPy arrays start with 0, meaning that the first element has index 0, and the second has index 1 etc.
- Example: to access 1-D array

```
import numpy as np
arr = np.array([1, 2, 3, 4])
print(arr[0])
print(arr[1])
```

```
print(arr[2] + arr[3])
```

### Access 2-D Arrays

To access elements from 2-D arrays we can use comma separated integers representing the dimension and the index of the element.

```
import numpy as np
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])
print('2nd element on 1st row: ', arr[0, 1])  #[2]
print('5th element on 2nd row: ', arr[1, 4])  #[10]
```

### Access 3-D Arrays

To access elements from 3-D arrays we can use comma separated integers representing the dimensions and the index of the element.

```
import numpy as np

arr = np.array([[[1, 2, 3], [4, 5, 6]],
                [[7, 8, 9], [10, 11, 12]]])

print(arr[0, 1, 2])  # 6
```

### Slicing on ndarray:

#### Syntax:

**Ndarrayobject[start:stop:step]**

Example:

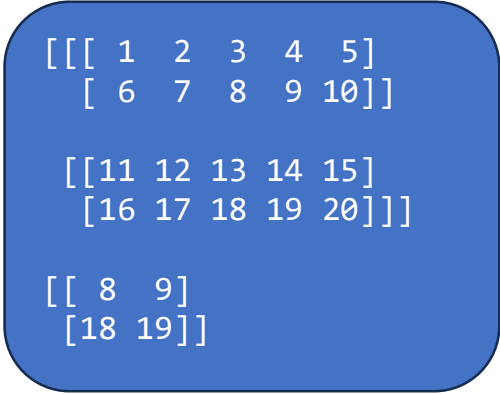
```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[1:5])  #[2,3,4,5]
print(arr[4:])  # [5,6,7]
print(arr[:4])  #[0,1,2,3]
print(arr[-3:-1]) # negative indexing is also possible [5,6]
```

**Slicing 2-D array:**

```
import numpy as np
arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
# From second element, slice 2nd to 4th element
print(arr[1, 1:4]) #[7,8,9]
# From both elements, get the 3rd element
print(arr[0:2, 2]) #[3 8]
# From both elements, slice index 1 to index 4 (not included), this will return a 2-D array:
print(arr[0:2, 1:4]) # [[2 3 4]
                    [7 8 9]]
```

**Slicing 3-D array:**

```
import numpy as np
arr = np.array([[[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]], [[11,12,13,14,15],[16,17,18,19,20]]])
print(arr)
# From both 2d array slice the element at index 1 and from that slice index 2 and index 3
print(arr[0:2, 1,2:4])
```



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

 [[11 12 13 14 15]
  [16 17 18 19 20]]

 [[ 8  9]
  [18 19]]
```

**Creating Numpy array from existing data (numpy.asarray())**

- NumPy provides us the way to create an array by using the existing data.
- **numpy.asarray** : This routine is used to create an array by using the existing data in the form of lists, or tuples. This routine is useful in the scenario where we need to convert a python sequence into the numpy array object.
- Syntax:  
`numpy.asarray(sequence, dtype = None, order = None)`

Where,

sequence: It is the python sequence which is to be converted into the python array.

dtype: It is the data type of each item of the array.

order: It can be set to C or F. The default is C.

Example:

```
import numpy as np
```

```
l=[1,2,3,4,5,6,7]
```

```
a = np.asarray(l);
```

```
print(type(a))
```

```
print(a)
```



```
<class 'numpy.ndarray'>
[1 2 3 4 5 6 7]
```

## NumPy Arrays Creation with Initial Placeholder Content

Often, the elements of an array are initially unknown, but its size is known. Hence, NumPy offers several functions to create arrays with initial placeholder content which are shown in the following table.

### NumPy Arrays Creation Functions

---

Function Name	Description
<code>np.zeros(shape)</code>	Creates an array of zeros
<code>np.ones(shape)</code>	Creates an array of ones
<code>np.empty(shape)</code>	Creates an empty array
<code>np.full(shape, fillvalue)</code>	Creates a full array
<code>np.eye(shape,k=0)</code>	Creates an identity matrix
<code>np.random.random()</code>	Creates an array with random values
<code>np.arange()</code>	The syntax for <code>arange()</code> is,  <i><code>np.arange([start,]stop, [step,][dtype=None])</code></i>

Returns evenly spaced values within a given interval where *start* (a number and optional) is the start of interval and its default value is zero, *stop* (a number) is the end of interval,

and *step* (a number and is optional) is the spacing between the values and *dtype* is the type of output array.

### **NumPy Array Manipulation Functions:**

**add():** This function is used to output the addition of two arrays.

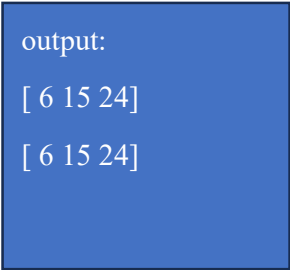
```
import numpy as np
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])
c = np.add(a,b)
print(c)
print(a + b)
```



```
output:
[5 7 9]
[5 7 9]
```

**subtract():** This function is used to output the subtraction of two arrays.

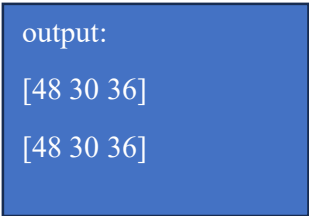
```
import numpy as np
a = np.array([10, 20, 30])
b = np.array([4, 5, 6])
c = np.subtract(a,b)
print(c)
print(a - b)
```



```
output:
[ 6 15 24]
[ 6 15 24]
```

**multiply():** This function is used to output the product of two arrays.

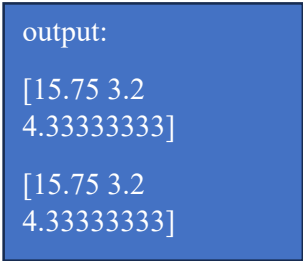
```
import numpy as np
a = np.array([12, 6, 6])
b = np.array([4, 5, 6])
c = np.multiply(a,b)
print(c)
print(a * b)
```



```
output:
[48 30 36]
[48 30 36]
```

**divide():** This function is used to output the division of two arrays

```
import numpy as np
a = np.array([63, 16, 26])
b = np.array([4, 5, 6])
c = np.divide(a,b)
print(c)
```



```
output:
[15.75 3.2
 4.33333333]
[15.75 3.2
 4.33333333]
```

```
print(a / b)
```

mod() and remainder(): These functions are used to output the remainder of two arrays.

```
import numpy as np
```

```
a = np.array([63, 16, 26])
```

```
b = np.array([4, 5, 6])
```

```
c = np.remainder(a,b)
```

```
d = np.mod(a,b)
```

```
print(c)
```

```
print(d)
```

output:

[3 1 2]

[3 1 2]

## **Pandas:**

- ☐ Pandas is a Python library used for working with data sets.
- ☐ It has functions for analyzing, cleaning, exploring, and manipulating data.
- ☐ The name "Pandas" has a reference to both "Panel Data", and "Python Data Analysis" and was created by Wes McKinney in 2008.

## **Uses of Pandas:**

- ☐ Pandas allow us to analyze big data and make conclusions based on statistical theories.
- ☐ Pandas can clean messy data sets, and make them readable and relevant.
- ☐ Relevant data is very important in data science.

## **Installing Pandas:**

- ☐ Pandas can be installed using the python package manger pip via command-line.
- ☐ The command for installing Pandas is:

**pip install pandas**

## **Series:**

- ☐ A Pandas Series is like a column in a table.
- ☐ It is a one-dimensional array holding data of homogeneous type.
- ☐ It can be created from a list as below

```
import pandas as pd
```

```
a = [1, 7, 2]
```



```
srs = pd.Series(a)
print(srs)
```

```
output:
0 1
1 7
2 2
dtype: int64
```

**DataFrames:**

- A Pandas DataFrame is a two-dimensional data structure, like a two-dimensional array, or a table with rows and columns.
- It can be created from a dictionary as below

```
import pandas as pd
data = {
    "calories": [420, 380, 390],
    "duration": [50, 40, 45]
}
df = pd.DataFrame(data)
print(df)
```

```
output:
   calories  duration
0     420         50
1     380         40
2     390         45
```