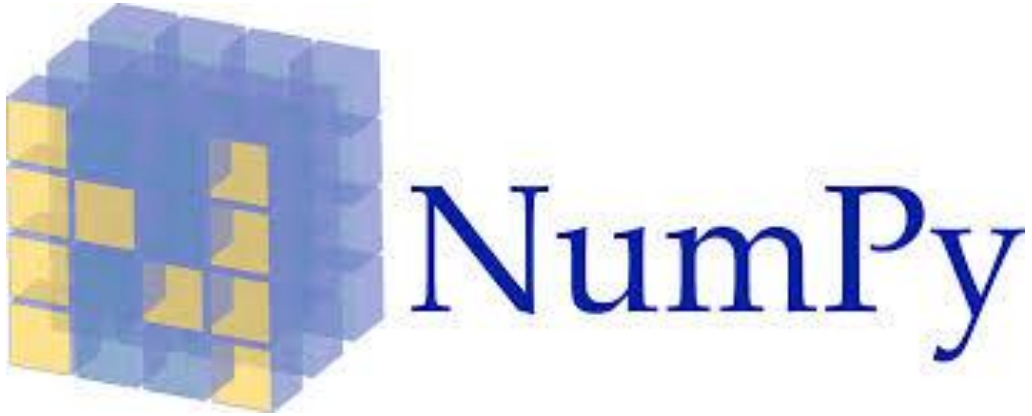
# Topic: PYTHON NUMPY

# Outline

- What is Python Numpy?
- How to install Numpy?
- Python Numpy Operation
- Python Numpy Special Functions

# What is Python NumPy?

- NumPy is a Python package which stands for ‘Numerical Python’.
- It is the core library for scientific computing, which contains a powerful n-dimensional array object.
- NumPy is a python library used for working with arrays.
- It is also useful in linear algebra, fourier transform, matrices etc.
- NumPy array can also be used as an efficient multi-dimensional.



# 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 that 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.

# NumPy Array

- Numpy array is a powerful n-dimensional array object which is in the form of rows and columns.
- It provides a high-performance multidimensional array object, and tools for working with these arrays.


Here, different elements are stored in their respective memory locations. It is said to be two dimensional because it has rows as well as columns. In the above image, we have 3 columns and 4 rows available.

# How do I install Numpy?

- To install Python NumPy, go to your command prompt and type "**pip install numpy**".
- Once the installation is completed, go to your IDE and simply import it by typing:

**"import numpy as np"**

# Numpy Array

- Single Dimensional Numpy Array

Example:

```
#Single-dimensional Numpy Array
import numpy as np
a=np.array([1,2,3])
print(a)
```

```
[1 2 3]
```

- Multi Dimensional Numpy Array

Example:

```
#Multi-dimensional Array
a=np.array([(1,2,3),(4,5,6)])
print(a)
```

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

# Python Numpy Operations

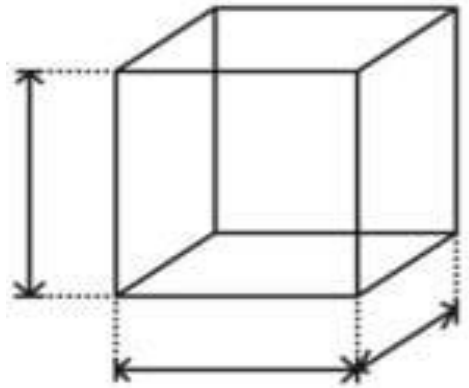
**ndim:**

- You can find the dimension of the array, whether it is a two-dimensional array or a single dimensional array.
- In the below code, with the help of ‘ndim’ function, we can find whether the array is of single dimension or multi dimension.

Example:

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

```
2
```



# Python Numpy Operations…

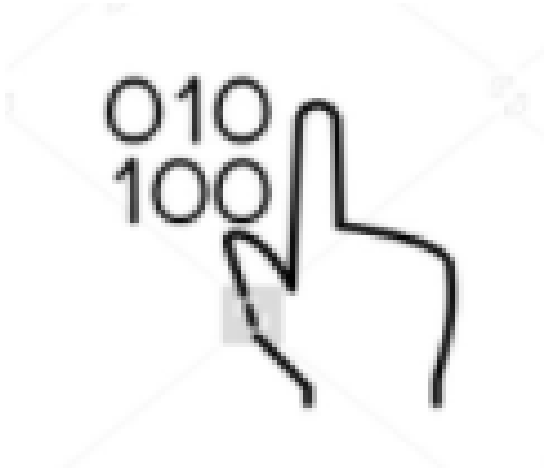
**itemsize:**

You can calculate the byte size of each element. In the below code, a single dimensional array is defined and with the help of ‘itemsize’ function, we can find the size of each element.

Example:

```
import numpy as np
a = np.array([(1,2,3)])
print(a.itemsize)
```

```
4
```



# Python Numpy Operations…

**dtype:**

You can find the data type of the elements that are stored in an array. So, if you want to know the data type of a particular element, you can use ‘dtype’ function which will print the datatype along with the size. In the below code, an array is defined the same function is used.

Example:

```
import numpy as np
a = np.array([(1,2,3)])
print(a.dtype)
```

```
int32
```

# Python Numpy Operations…

Similarly, to find the **size** and **shape** of the array using ‘size’ and ‘shape’ function respectively.

Example:

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

```
6
(1, 6)
```

# Python Numpy Operations…

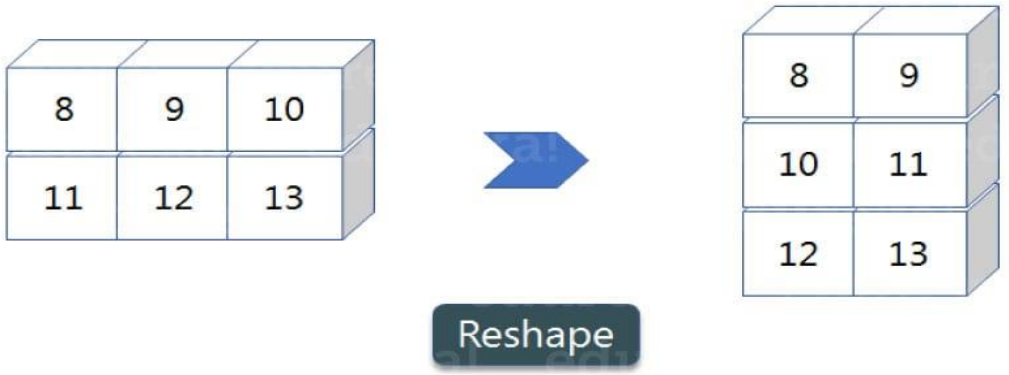
**reshape:**

Reshape is when you change the number of rows and columns which gives a new view to an object. Now, let us take an example to reshape the below array:

Example:

```
import numpy as np
a = np.array([(8,9,10),(11,12,13)])
print(a)
a=a.reshape(3,2)
print(a)
```

```
[[ 8  9 10]
 [11 12 13]]
[[ 8  9]
 [10 11]
 [12 13]]
```




As you can see in the above image, we have 3 columns and 2 rows which has converted into 2 columns and 3 rows.

# Python Numpy Operations…

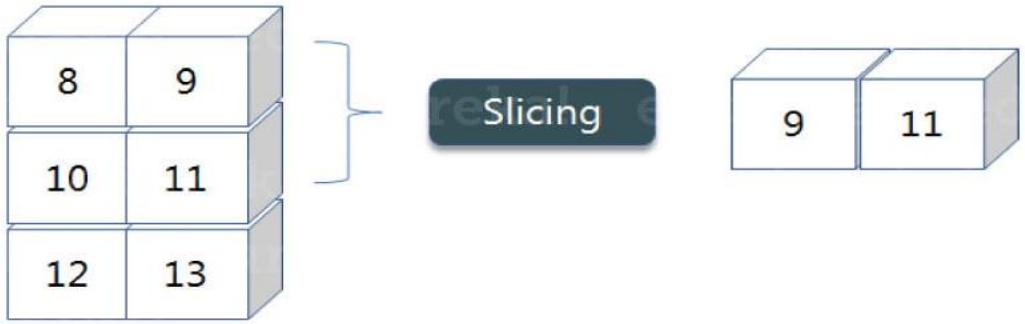
**slicing:**

Slicing is basically extracting particular set of elements from an array. This slicing operation is pretty much similar to the one which is there in the list as well. Consider the following example:

Example:

```
import numpy as np
a=np.array([(1,2,3,4),(3,4,5,6)])
print(a[0,2])
```

```
3
```



# Python Numpy Operations…

**linspace:**

This is another operation in python numpy which returns evenly spaced numbers over a specified interval. Consider the below example:

Example:

```
import numpy as np
a=np.linspace(1,3,10)
print(a)
```

```
[1.         1.22222222 1.44444444 1.66666667 1.88888889 2.11111111
 2.33333333 2.55555556 2.77777778 3.        ]
```

# Python Numpy Operations…

**max/min:**

Next, we have some more operations in numpy such as to find the minimum, maximum as well the sum of the numpy array.

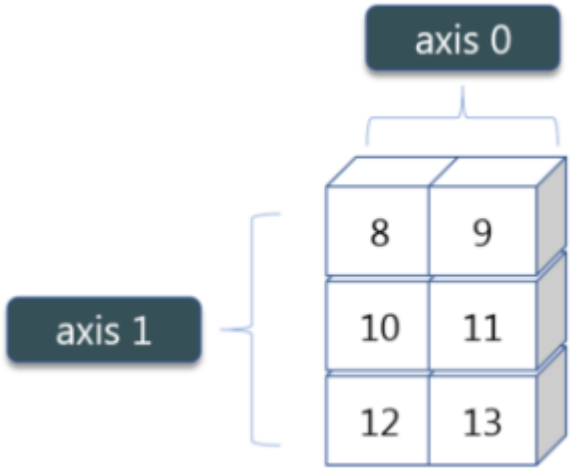
Example:

```
import numpy as np

a= np.array([1,2,3])
print(a.min())
print(a.max())
print(a.sum())
```

```
1
3
6
```

# The concept of axis in Python Numpy




Here the rows are called as axis 1 and the columns are called as axis 0. Now you must be wondering what is the use of these axis?
Suppose you want to calculate the sum of all the columns, then you can make use of axis.

Example:

```
a= np.array([(1,2,3),(3,4,5)])
print(a.sum(axis=0))
```

```
[4 6 8]
```

# Python Numpy Operations…

**Square Root & Standard deviation:**

There are various mathematical functions that can be performed using python numpy. We can find the square root, standard deviation of the array.

Example:

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

```
[[1.         1.41421356 1.73205081]
 [1.73205081 2.         2.23606798]]
1.2909944487358056
```

# Python Numpy Operations…

**Addition Operation:**

More operations can be performed on numpy array i.e addition, subtraction, multiplication and division of the two matrices.

Example:

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

```
[[ 2  4  6]
 [ 6  8 10]]
```

Similarly, we can perform other operations such as subtraction, multiplication and division.

Example:

```
import numpy as np
x= np.array([(1,2,3),(3,4,5)])
y= np.array([(1,2,3),(3,4,5)])
print(x-y)
print(x*y)
print(x/y)
```

```
[[0 0 0]
 [0 0 0]]
[[ 1  4  9]
 [ 9 16 25]]
[[1. 1. 1.]
 [1. 1. 1.]]
```

# Python Numpy Operations…

**Vertical & Horizontal Stacking:**

Next, if you want to concatenate two arrays and not just add them, you can perform it using two ways – *vertical stacking* and *horizontal stacking*.

Example:

```
import numpy as np
x= np.array([(1,2,3),(3,4,5)])
y= np.array([(1,2,3),(3,4,5)])
print(np.vstack((x,y)))
print(np.hstack((x,y)))
```

```
[[1 2 3]
 [3 4 5]
 [1 2 3]
 [3 4 5]]
[[1 2 3 1 2 3]
 [3 4 5 3 4 5]]
```

# Python Numpy Operations…

**ravel:**

There is one more operation where you can convert one numpy array into a single column i.e *ravel*.

Example:

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

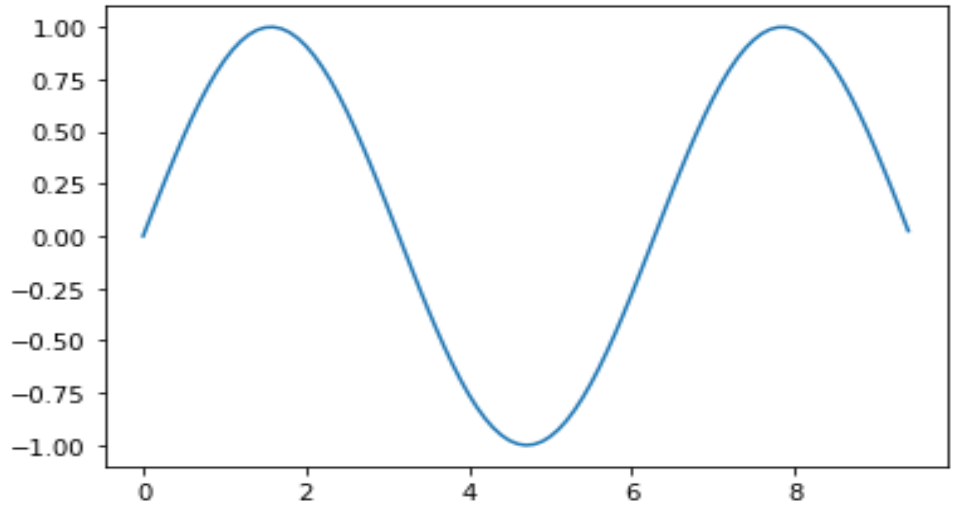
```
[1 2 3 3 4 5]
```

# Python Numpy Special Functions

There are various special functions available in numpy such as sine, cosine, tan, log etc. First, let's begin with sine function. For that, we need to import a module called *matplotlib*

Example:

```
import numpy as np
import matplotlib.pyplot as plt
x= np.arange(0,3*np.pi,0.1)
y=np.sin(x)
plt.plot(x,y)
plt.show()
```

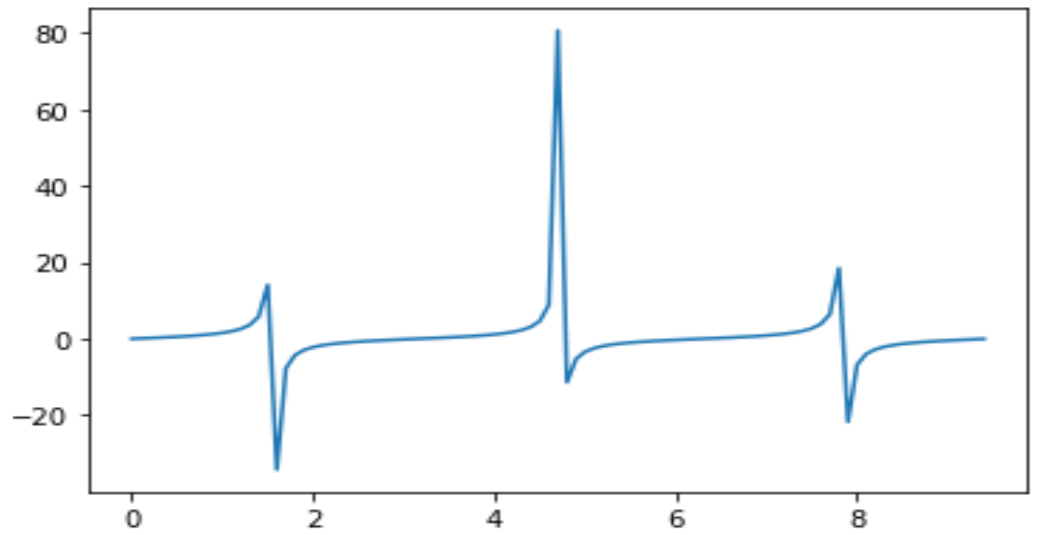


# Python Numpy Operations…

Similarly, you can plot a graph for any trigonometric function such as cos, tan etc. below is a example where you can plot a graph of another function, i.e. ***tan***.

Example:

```
import numpy as np
import matplotlib.pyplot as plt
x= np.arange(0,3*np.pi,0.1)
y=np.tan(x)
plt.plot(x,y)
plt.show()
```



# Python Numpy Operations…

Some other special functionality in numpy array such as exponential and logarithmic function. Now in exponential, the *e* value is somewhere equal to 2.7 and in log, it is actually *log base 10*. When we discuss about natural log i.e log base e, it is referred as Ln.

Example:

```
a= np.array([1,2,3])
print(np.exp(a))
```

```
[ 2.71828183  7.3890561  20.08553692]
```

As you can see the above output, the exponential values are printed i.e *e* raise to the power 1 is *e*, which gives the result as 2.718… Similarly, *e* raise to the power of 2 gives the value somewhere near 7.38 and so on.

# Python Numpy Operations…

Next, in order to calculate log, let's see how you can implement it:

Example:

```
import numpy as np
import matplotlib.pyplot as plt
a= np.array([1,2,3])
print(np.log(a))
```

```
[0.         0.69314718 1.09861229]
```

Here, we have calculated natural log which gives the value as displayed above. Now, if we want log base 10 instead of Ln or natural log, you can follow the below code:

```
import numpy as np
import matplotlib.pyplot as plt
a= np.array([1,2,3])
print(np.log10(a))
```

```
[0.         0.30103    0.47712125]
```

# THANKYOU