NumPy

- NumPy stands for Numerical Python.
- NumPy is a Python library used for working with arrays.
- NumPy was created in 2005 by Travis Oliphant.
- It is an open source and you can use it freely.

Uses of NumPy

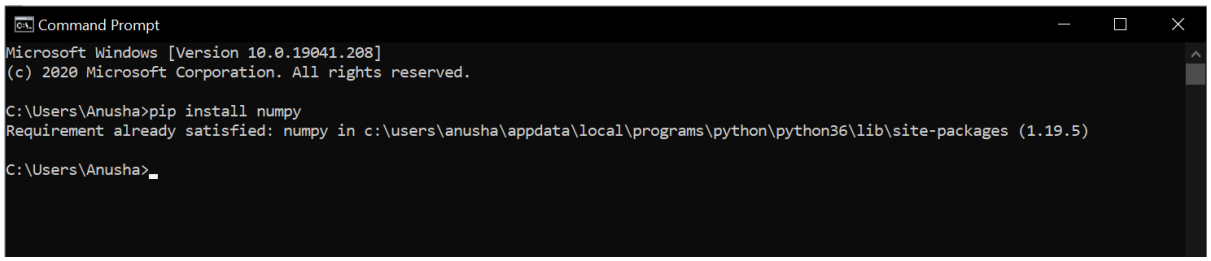
- In Python we have lists that serve the purpose of arrays, but they are slow to process.
- NumPy aims to provide an array object that is up to 50x faster than traditional Python lists.

- The array object in NumPy is called `ndarray`, it provides a lot of supporting functions that make working with `ndarray` very easy.

NumPy Faster Than Lists - NumPy arrays are stored at one continuous place in memory unlike lists, so processes can access and manipulate them very efficiently.

NumPy installation- If you have Python and PIP already installed on a system, then installation of NumPy is very easy.

Step1: - Install it using this command: ***pip install numpy***



Step2: - Once NumPy is installed, import it in your applications by adding the `import` keyword: ***import numpy***

Example1:

```
>>>
>>> import numpy
>>> arr=numpy.array([10,20,30,40,50,60])
>>> print(arr)
[10 20 30 40 50 60]
>>>
```

Example2:

***NumPy as np (*alias:** An alternate name for referring to the same thing.**)**

__version__ :provides numpy version.

```
>>> import numpy as np
>>> arr=np.array([100,200,300,400,500,600])
>>> print(arr)  ; print(np.__version__)
[100 200 300 400 500 600]
1.19.5
```

Example3:

```
>>> import numpy as np
>>> arr = np.array([1, 2, 3, 4, 5])
>>> print(arr)
[1 2 3 4 5]
>>> print(type(arr))
<class 'numpy.ndarray'>
>>>
```

Example4: **Use a tuple to create a NumPy array**

```
>>> import numpy as np
>>> arr = np.array((100, 200, 300, 400, 500))
>>> print(arr)
[100 200 300 400 500]
```

Dimensions in Arrays

Example1:

```
import numpy as np        #alias
#0-D array
arr = np.array(24) ; print("0-D array")  ; print(arr)

#1-D array
print("1-D array") ; arr = np.array([24,33,42,51,60,78,87,96,105])    ; print(arr)

#2-D array
print("2-D array") ; arr = np.array([[24,33,42] , [51,60,78]])    ; print(arr)

#3-D arrary
print("3-D array") ; arr = np.array([[[24,33,42] , [51,60,78]] , [[87,96,105] ,[114,123,132]]])    ; print(arr)
```

Example2: Check how many dimensions the arrays have:

```
import numpy as np        #alias
a= np.array(24)
b = np.array([24,33,42,51,60,78,87,96,105])
c= np.array([[24,33,42] , [51,60,78]])
d= np.array([[[24,33,42] , [51,60,78]] , [[87,96,105] ,[114,123,132]]])
#Check Number of Dimensions - ndim
print(a.ndim)
print(b.ndim)
print(c.ndim)
print(d.ndim)
```

Access Array Elements

```
import numpy as np        #alias
q = np.array([15,24,33,42,51,60,78,87,96,105])
#Accessing the array element
print(q[1])
print("adding 2 index value -") ; print(q[3] + q[5])

#Access 2-D array
w= np.array([[15,24,33,42,96] , [51,60,78,87,105]])
print('2nd element on 1st row: ', w[0, 1])
print('5th element on 2nd row: ', w[1, 4])

#Access 3-D array
t = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
print("3-D array") ; print(t[0, 1, 2])
print(t[1, 0, 1])

#Negative indexing -2dim
n = np.array([[1,2,3,4,5], [6,7,8,9,10]])
print('Last element from 2nd dim: ', n[1, -1])
```

NumPy Array Slicing

```
import numpy as np        #alias
q = np.array([15,24,33,42,51,60,78,87,96,105])
#Slicing arrays [start:end]
print(q[1:6])
print(q[5:])
print(q[:5])
print(q[-3:-1])

#Slicing arrays [start:end:step]
print(q[1:9:2])
print(q[::2])

#Slicing 2-D array
z = np.array([[15,24,33,42,51] ,[60,78,87,96,105]])
print(z[1, 1:5])
print(z[0:4, 2])     #From both elements, return index 2
print(z[0:4, 1:6])
```

The Difference Between Copy and View

The main difference between a copy and a view of an array is that the copy is a new array, and the view is just a view of the original array.

The ***copy*** *owns* the data and any changes made to the copy will not affect original array, and any changes made to the original array will not affect the copy.

The ***view*** *does not own* the data and any changes made to the view will affect the original array, and any changes made to the original array will affect the view.

Example: Copy Vs View

```
import numpy as np
#copy
arr = np.array([1, 2, 3, 4, 5])
x = arr.copy()
arr[0] = 42
print(arr) ; print(x)

#view
a = np.array([11, 12, 31, 14, 51])
q = a.view()
a[0] = 42
print() ; print(a) ; print(q)
```

```
[42  2  3  4  5]
[1 2 3 4 5]

[42 12 31 14 51]
[42 12 31 14 51]
>
```

NumPy Sorting Arrays

Sorting means putting elements in an *ordered sequence*. ***Ordered sequence*** is any sequence that has an order corresponding to elements, like numeric or alphabetical, ascending or descending.

Example: sort()

```
import numpy as np
a = np.array([24,15,6,33, 105, 42, 60, 51])
#Sort
print(np.sort(a))
#Sort the array alphabetically
f = np.array(['sweet lime','mango','kiwi','orange','pineapple'
    ,'banana', 'cherry', 'apple'])
print(np.sort(f))

#2-D sort
d = np.array([[33,105 ,51,24, 42], [96,51,123, 60,78 ]])
print(np.sort(d))
```

```
[  6  15  24  33  42  51  60 105]
['apple' 'banana' 'cherry' 'kiwi' 'mango' 'orange' 'pineapple'
 'sweet lime']
[[ 24  33  42  51 105]
 [ 51  60  78  96 123]]
>
```