Numpy Installation

[ ]:
```
!pip install numpy
```

[38]:
```
import numpy as np
```

Arrays

How are numpy array faster than list ?

- Numpy arrays are basically stored sequentially in the memory.

[39]:
```
arr = np.array([1, 2, 3, 4, 5])
arr = np.array((1, 2, 3, 4, 5))
print(arr)
print(type(arr))
```

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

[40]:
```
# 2D ARRAYS
arr = np.array([[1, 2, 3], [4, 5, 6]])

print(arr)
```

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

[41]:
```
# 3D ARRAYS
arr = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])

print(arr)
```

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

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

[42]:
```
#dimension of the array
arr.ndim
```

[42]:
```
3
```

[43]:
```
# indexing 1D
arr = np.array([1, 2, 3, 4])
arr[0]
```

[43]:
```
1
```

[44]:
```
# indexing 2d
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])

print(' 0th row and 1st value :  ', arr[0, 1])
```

```
 0th row and 1st value :   2
```

[45]:
```
# Indexing in 3D Arrays
arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])

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

```
6
```

[46]:
```
#slicing 1D
arr = np.array([1, 2, 3, 4, 5, 6, 7])

print(arr[1:5])
```

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

[47]:
```
# Negative Slicing 1D
arr = np.array([1, 2, 3, 4, 5, 6, 7])

print(arr[-3:-1])
```

```
[5 6]
```

[48]:
```
# Slicing in steps 1D
arr = np.array([1, 2, 3, 4, 5, 6, 7])

# Prints from index 1 to 5 leaving 2 elements leaving an element in between
print(arr[1:5:2])
```

```
[2 4]
```

[49]:
```
# Slicing in steps  1D
arr = np.array([1, 2, 3, 4, 5, 6, 7])

print(arr[::2])
```

```
[1 3 5 7]
```

[50]:
```
#slicing 2D
arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
print(arr[1, 1:4])
```

```
[7 8 9]
```

Data types in Numpy

- `i` - integer
- `b` - boolean
- `u` - unsigned integer
- `f` - float
- `c` - complex float
- `m` - timedelta
- `M` - datetime
- `O` - object
- `S` - string
- `U` - unicode string
- `V` - fixed chunk of memory for other type (void)

[56]:
```
# array types
arr = np.array([1, 2, 3, 4])
print(type(arr))
print(type(arr[0]))
print(arr.dtype)
```

```
<class 'numpy.ndarray'>
<class 'numpy.int32'>
int32
```

[62]:
```
# each element is  a byte string
arr = np.array([1, 2, 3, 4], dtype='S')

print(arr)
print(arr.dtype) # represents each element is a byte string of length 1
```

```
[b'1' b'2' b'3' b'4']
|S1
```

[64]:
```
#data type 4 bytes integer
arr = np.array([1, 2, 3, 4], dtype='i4')

print(arr)
print(arr.dtype)
```

```
[1 2 3 4]
int32
```

[66]:
```
#converting data types
arr = np.array([1.1, 2.1, 3.1])

newarr = arr.astype('i')

print(newarr)
print(newarr.dtype)
```

```
[1 2 3]
int32
```

[67]:
```
#Numpy Copy Vs View

# Create an original array
original_array = np.array([1, 2, 3, 4, 5])

# Create a copy and a view of the original array
copied_array = original_array.copy()
view_array = original_array.view()

# Change an element in the original array
original_array[0] = 10

# Print arrays
print("Original Array:", original_array)
print("Copied Array:", copied_array)
print("View Array:", view_array)
```

```
Original Array: [10  2  3  4  5]
Copied Array: [1 2 3 4 5]
View Array: [10  2  3  4  5]
```

[75]:
```
#shape of the array
arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])

print("Shape : ",arr.shape)
print("Dimension :",arr.ndim)
```

```
Shape :  (2, 4)
Dimension : 2
```

[77]:
```
# reshape function
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])

newarr = arr.reshape(4, 3)

print(newarr)
```

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

[82]:
```
# Iterating a 1D Array
arr = np.array([1, 2, 3])
print("1D Array : ")
for x in arr:
  print(x)

print("2D Array: ")
# Iterating a 2D array
arr = np.array([[1, 2, 3], [4, 5, 6]])

for x in arr:
  print(x)

print("2D Array: ")
for x in arr:
  for y in x:
    print(y)

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

for x in arr:
  for y in x:
    for z in y:
      print(z)
```

```
1D Array :
1
2
3
2D Array:
[1 2 3]
[4 5 6]
2D Array:
1
2
3
4
5
6
3D Array:
1
```

```
2
3
4
5
6
7
8
9
10
11
12
```

[90]:
```
# nditer Function - we need to use n for loops which can be difficult to write for arrays with very high dimensionality so this can be used
arr = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])

for x in np.nditer(arr):
  print(x)
```

```
1
2
3
4
5
6
7
8
```

[91]:
```
#ndenumerate
arr = np.array([1, 2, 3])

for idx, x in np.ndenumerate(arr):
  print(idx, x)
```

```
(0,) 1
(1,) 2
(2,) 3
```

[92]:
```
# Concatenate
arr1 = np.array([1, 2, 3])

arr2 = np.array([4, 5, 6])

arr = np.concatenate((arr1, arr2))

print(arr)
```

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

[95]:
```
# Join two 2-D arrays along rows (axis=1):
arr1 = np.array([[1, 2], [3, 4]])

arr2 = np.array([[5, 6], [7, 8]])

arr = np.concatenate((arr1, arr2), axis=1)

print(arr)
```

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

[96]:
```
# Join two 2-D arrays along cols (axis=0):
arr1 = np.array([[1, 2], [3, 4]])

arr2 = np.array([[5, 6], [7, 8]])

arr = np.concatenate((arr1, arr2), axis=1)

print(arr)
```

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

[100]:
```
#split
arr = np.array([1, 2, 3, 4, 5, 6])

newarr = np.array_split(arr, 3)

print(newarr)
```

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

[103]:
```
# search
arr = np.array([1, 2, 3, 4, 5, 4, 4])

x = np.where(arr == 4)

print(x)
```

```
(array([3, 5, 6], dtype=int64),)
```

[104]:
```
# search even value indexes
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])

x = np.where(arr%2 == 0)

print(x)
```

```
(array([1, 3, 5, 7], dtype=int64),)
```

[109]:
```
#sorting an array
arr = np.array([3, 2, 0, 1])

print(np.sort(arr))
```

```
[0 1 2 3]
```

[111]:
```
# sort an bool array
arr = np.array([True, False, True])

print(np.sort(arr))
```

```
[False  True  True]
```

[113]:
```
# Array Filter
arr = np.array([41, 42, 43, 44])

x = [True, False, True, False]

newarr = arr[x]

print(newarr)
```

```
[41 43]
```

[118]:
```
#filter array more than 42
arr = np.array([41, 42, 43, 44])
```

```
filter_arr = arr > 42

newarr = arr[filter_arr]

print(filter_arr)
print(newarr)
```

```
[False False  True  True]
[43 44]
```

[117]:
```
# filter to have even numbers
arr = np.array([1, 2, 3, 4, 5, 6, 7])

filter_arr = arr % 2 == 0

newarr = arr[filter_arr]

print(filter_arr)
print(newarr)
```

```
[False  True False  True False  True False]
[2 4 6]
```

[ ]: