Basics of NumPy Arrays

In [ ]:

```
import numpy
numpy.__version__
```

NumPy (Numeric Python) provides an efficient way of storage and accessing the elements in numeric formats. They provide an easy to use interface by which the elements can be stored in a dense data structure and can be efficiently accessed whenever required.

Note - numpy is already installed when you are using Anaconda navigator.

In [ ]:

```
import numpy as np
```

Usually alias 'np' is used to refer the numpy library.

Python Arrays

Python has arrays that stores the data in efficient manner which is having a fixed data type. The built-in array module helps in doing so.

In [1]:

```
import array
my_array = array.array('i', [1, 2, 3, 4])

print("type   : ", type(my_array))
my_array
```

```
type   :  <class 'array.array'>
```

Out[1]:

```
array('i', [1, 2, 3, 4])
```

The above code snippet creates a array holding all the elements which are of same type i.e. integer. 'i', designates interger.

In [2]:

```
my_float_array = array.array('f', [1.1, 2.3, 4.5])
print("type   : ", type(my_float_array))
my_float_array
```

```
type   :  <class 'array.array'>
```

Out[2]:

```
array('f', [1.100000023841858, 2.299999952316284, 4.5])
```

The above code snippet creates a array holding all the elements which are of same type i.e. float. 'f', designates float or real number.

NumPy Arrays

But ndarray provides many operations on top of the arrays and hence needs to preferred over the built-in array module.

NumPy array creation from List

In [4]:

```
import numpy as np
```

Use np.array to create the ndarrays from Lists

In [5]:

```
my_list = [1, 2, 3, 4, 5]
my_np_array = np.array(my_list)
print("type   : ", type(my_np_array))
my_np_array
```

```
type   :  <class 'numpy.ndarray'>
```

Out[5]:

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

In [ ]:

```
my_float_list = [1.1, 2.2, 3.3]
my_float_array = np.array(my_float_list)
my_float_array
```

All elements should be of same type. If types do not match, Numpy will upcast if possible.

In [ ]:

```
my_mixed_list = [1.2, 1, 2, 3.4]
my_array = np.array(my_mixed_list)
my_array
```

Explicit type also can set for the array.

In [ ]:

```
my_list = [1, 2, 3, 4, 5]
my_array = np.array(my_list, dtype="int32")
my_array
```

In [ ]:

```
my_float_list = [1.1, 2.2, 3.3]
my_array = np.array(my_list, dtype="float32")
my_array
```

Array can be converted back to the list.

In [ ]:

```
my_list = [1, 2, 3, 4, 5]
my_array = np.array(my_list, dtype="int32")

back_to_list = my_array.tolist()
back_to_list
```

Array Attributes

In [ ]:

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

In [ ]:

```
#Data type of array
x.dtype
```

In [ ]:

```
#Number of dimensions i.e. ndim
x.ndim
```

In [ ]:

```
#Shape of array i.e. rows , column information
x.shape
```

In [ ]:

```
#Number of elements in array i.e. size = rows * columns
x.size
```

Creating Arrays filled in with default values

In many cases, it will be required to use the arrays which has some default elements present inside it like ones or zeros or some fixed number. NumPy provides built-in functions to create such arrays directly.

In [ ]:

```
#Create array filled of 5 ones
np.ones(5)
```

In [ ]:

```
#Create array filled of 5 ones which are integers
np.ones(5, dtype="int")
```

In [ ]:

```
#Create array filled of 5 zeros
np.zeros(5)
```

In [ ]:

```
#Create array filled of 5 zeros which are integers
np.zeros(5, dtype=int)
```

In [ ]:

```
#Create uninitialized array of integers, any garbage value will appear in it
np.empty(5)
```

Creating linear sequence arrays

In [ ]:

```
#Create an array having element range between 0 to 10 spaced at distance of 1
np.arange(0, 10)
```

In [ ]:

```
#Create an array having element range between 0 to 10 spaced at distance of 2
np.arange(0, 10, 2)
```

In [ ]:

```
#Create an array having element range between 5 to 15 spaced at distance of 3
np.arange(5, 15, 3)
```

In [ ]:

```
#Create an array having element range between 0 to 1 having 5 values which are equidistant
np.linspace(0, 1, 5)
```

In [ ]:

```
#Create an array having element range between 10 to 20 having 5 values which are equidistan
np.linspace(10, 20, 5)
```

Creating multi-dimensional arrays

In [ ]:

```
#Create a two dimensional array with lists as its rows
list1 = [1, 2, 3]
list2 = [4, 5, 6]
list3 = [7, 8, 9]
my_nd_array = np.array([list1, list2, list3])
my_nd_array
```

In [ ]:

```
#Number of dimensions
my_nd_array.ndim
```

In [ ]:

```
#Shape of array
my_nd_array.shape
```

In [ ]:

```
#Size of array  rows * columns
my_nd_array.size
```

In [ ]:

```
#Create a 2-d array with dimensions 3 * 5 filled with all zeros
np.zeros((3,5))
```

In [ ]:

```
#Create a 2-d array with dimenstions 4 * 4 filled with all ones
np.ones((4,4))
```

In [ ]:

```
#Create identity matrix of dimensions 4 * 4
np.eye(4)
```

Exercise

Q1. Create an one dimensional array filled with random numbers. Write a function that prints out the maximum and minimum values from the array

In [ ]:

```
#Try it here :
```

Q2. Create an one dimensional array filled with random numbers. Write a function that returns an array filled with reciprocal of array elements.

In [ ]:

```
#Try it here :
```

Q3. Write a function that accepts one dimensional array as input and another number. The funtion should return the array containing all the elements which are greater than the number given as input to the function.

In [ ]:

```
#Try it here :
```