NumPy

1. For importing numpy

```
import numpy as np
```

2.Creating different types of numpy arrays

```
# Defining 1D array
my1DArray = np.array([1, 8, 27, 64])
print(my1DArray)
```

```
[ 1  8 27 64]
```

```
# Defining and printing 2D array
my2DArray = np.array([[1, 2, 3, 4], [2, 4, 9, 16], [4, 8, 18, 32]])
print(my2DArray)
```

```
[[ 1  2  3  4]
 [ 2  4  9 16]
 [ 4  8 18 32]]
```

```
#Defining and printing 3D array
my3Darray = np.array([[[ 1, 2 , 3 , 4],[ 5 , 6 , 7 ,8]], [[ 1, 2, 3,
4],[ 9, 10, 11, 12]]])
print(my3Darray)
```

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

 [[ 1  2  3  4]
  [ 9 10 11 12]]]
```

3. Basic information, such as the data type, shape, size, and strides of a NumPy array

```
# Print out memory address
print(my2DArray.data)
```

```
<memory at 0x7fbe7448b440>
```

```
# Print the shape of array
print(my2DArray.shape)
```

```
(3, 4)
```

```
# Print out the data type of the array
print(my2DArray.dtype)
```

```
int64
```

```
# Print the stride of the array.
print(my2DArray.strides)
```

```
(32, 8)
```

4. creating an array using built-in NumPy functions

```
# Array of ones
ones = np.ones((3,4))
print(ones)
```

```
[[1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]]
```

```
# Array of zeros
zeros = np.zeros((2,3,4),dtype=np.int16)
import numpy as np
#zeros = np.zeros((2,3,4))
print(zeros)
```

```
[[[0 0 0 0]
  [0 0 0 0]
  [0 0 0 0]]

 [[0 0 0 0]
  [0 0 0 0]
  [0 0 0 0]]]
```

```
# Array with random values
np.random.random((2,2))
```

```
array([[0.09061085, 0.52169507],
       [0.83466029, 0.92592689]])
```

```
# Empty array
emptyArray = np.empty((3,2))
print(emptyArray)
```

```
[[2.07191844e-316 0.00000000e+000]
 [0.00000000e+000 0.00000000e+000]
 [0.00000000e+000 0.00000000e+000]]
```

```
# Full array
fullArray = np.full((2,2),7)
print(fullArray)
```

```
[[7 7]
 [7 7]]
```

```
# Array of evenly-spaced values
evenSpacedArray = np.arange(10,25,5)
print(evenSpacedArray)
```

```
[10 15 20]
```

```
# Array of evenly-spaced values
evenSpacedArray2 = np.linspace(0,2,9)
print(evenSpacedArray2)
```

```
[0.   0.25 0.5  0.75 1.   1.25 1.5  1.75 2.  ]
```

For NumPy arrays and file operations, we will use the following code:

```
# Save a numpy array into file
x = np.arange(0.0,50.0,1.0)
np.savetxt('data.out', x, delimiter=',')
# Loading numpy array from text
z = np.loadtxt('data.out', unpack=True)
print(z)
```

```
[ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10. 11. 12. 13. 14. 15. 16. 17.
 18. 19. 20. 21. 22. 23. 24. 25. 26. 27. 28. 29. 30. 31. 32. 33. 34. 35.
```

```
     36. 37. 38. 39. 40. 41. 42. 43. 44. 45. 46. 47. 48. 49.]
```

```
# Loading numpy array using genfromtxt method
my_array2 = np.genfromtxt('data.out',  skip_header=1, filling_values=-999)
print(my_array2)
# For inspecting NumPy arrays, we will use the following code:
# Print the number of `my2DArray`'s dimensions
print(my2DArray.ndim)
# Print the number of `my2DArray`'s elements
print(my2DArray.size)
# Print information about `my2DArray`'s memory layout
print(my2DArray.flags)
# Print the length of one array element in bytes
print(my2DArray.itemsize)
# Print the total consumed bytes by `my2DArray`'s elements
print(my2DArray.nbytes)
```

```
[ 1.  2.  3.  4.  5.  6.  7.  8.  9. 10. 11. 12. 13. 14. 15. 16. 17. 18.
 19. 20. 21. 22. 23. 24. 25. 26. 27. 28. 29. 30. 31. 32. 33. 34. 35. 36.
 37. 38. 39. 40. 41. 42. 43. 44. 45. 46. 47. 48. 49.]
2
12
  C_CONTIGUOUS : True
  F_CONTIGUOUS : False
  OWNDATA : True
  WRITEABLE : True
  ALIGNED : True
  WRITEBACKIFCOPY : False
  UPDATEIFCOPY : False

8
96
```

Broadcasting is a mechanism that permits NumPy to operate with arrays of different shapes when performing arithmetic operations:

```
# Rule 1: Two dimensions are operatable if they are equal
# Create an array of two dimension
A =np.ones((6, 8))
# Shape of A
print(A.shape)
# Create another array
B = np.random.random((6,8))
# Shape of B
print(B.shape)
# Sum of A and B, here the shape of both the matrix is same.
print(A + B)
```

```
(6, 8)
(6, 8)
[[1.28661106 1.92031694 1.38014198 1.09645134 1.07508983 1.16207631
  1.05273306 1.4968388 ]
 [1.56609189 1.50845995 1.67754456 1.32829112 1.85846453 1.73551059
  1.25671481 1.24048294]
 [1.479519   1.90921077 1.89430046 1.00268604 1.51805357 1.80340759
  1.53234763 1.2416502 ]
 [1.71719771 1.35590155 1.62439789 1.74101769 1.13063742 1.40671043
  1.96086909 1.62825653]
 [1.45329399 1.17890348 1.66144383 1.60555408 1.24929072 1.10274244
  1.64363444 1.94106252]
 [1.32208184 1.48306746 1.92185948 1.5293374  1.03581895 1.94472442
  1.32234746 1.99539484]]
```

Secondly, two dimensions are also compatible when one of the dimensions of the array is 1. Check the example given here:

```
# Rule 2: Two dimensions are also compatible when one of them is 1
# Initialize `x`
x = np.ones((3,4))
print(x)
# Check shape of `x`
print(x.shape)
# Initialize `y`
y = np.arange(4)
print(y)
# Check shape of `y`
print(y.shape)
```

```
# Subtract `x` and `y`
print(x - y)
```

```
[[1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]]
(3, 4)
[0 1 2 3]
(4,)
[[ 1.  0. -1. -2.]
 [ 1.  0. -1. -2.]
 [ 1.  0. -1. -2.]]
```

Lastly, there is a third rule that says two arrays can be broadcast together if they are compatible in all of the dimensions. Check the example given here:

```
# Rule 3: Arrays can be broadcast together if they are compatible in all dimensions
x = np.ones((6,8))
y = np.random.random((10, 1, 8))
print(x + y)
```

```
  [1.69979441 1.10754371 1.5721646  1.9263917  1.86077769 1.68613724
   1.39158563 1.81775193]
  [1.69979441 1.10754371 1.5721646  1.9263917  1.86077769 1.68613724
   1.39158563 1.81775193]
  [1.69979441 1.10754371 1.5721646  1.9263917  1.86077769 1.68613724
   1.39158563 1.81775193]]

 [[1.02280244 1.94494069 1.50566405 1.87563941 1.12227536 1.54905491
   1.5197974  1.97542903]
  [1.02280244 1.94494069 1.50566405 1.87563941 1.12227536 1.54905491
   1.5197974  1.97542903]
  [1.02280244 1.94494069 1.50566405 1.87563941 1.12227536 1.54905491
   1.5197974  1.97542903]
  [1.02280244 1.94494069 1.50566405 1.87563941 1.12227536 1.54905491
   1.5197974  1.97542903]
  [1.02280244 1.94494069 1.50566405 1.87563941 1.12227536 1.54905491
   1.5197974  1.97542903]
  [1.02280244 1.94494069 1.50566405 1.87563941 1.12227536 1.54905491
   1.5197974  1.97542903]]

 [[1.87543015 1.90066496 1.85452774 1.35655974 1.91343358 1.37467132
   1.15842006 1.25456455]
  [1.87543015 1.90066496 1.85452774 1.35655974 1.91343358 1.37467132
   1.15842006 1.25456455]
  [1.87543015 1.90066496 1.85452774 1.35655974 1.91343358 1.37467132
   1.15842006 1.25456455]
  [1.87543015 1.90066496 1.85452774 1.35655974 1.91343358 1.37467132
   1.15842006 1.25456455]
  [1.87543015 1.90066496 1.85452774 1.35655974 1.91343358 1.37467132
   1.15842006 1.25456455]
  [1.87543015 1.90066496 1.85452774 1.35655974 1.91343358 1.37467132
   1.15842006 1.25456455]]

 [[1.66075959 1.14041294 1.21459967 1.57492004 1.30070419 1.80427247
   1.68929017 1.3232186 ]
  [1.66075959 1.14041294 1.21459967 1.57492004 1.30070419 1.80427247
   1.68929017 1.3232186 ]
  [1.66075959 1.14041294 1.21459967 1.57492004 1.30070419 1.80427247
   1.68929017 1.3232186 ]
  [1.66075959 1.14041294 1.21459967 1.57492004 1.30070419 1.80427247
   1.68929017 1.3232186 ]
  [1.66075959 1.14041294 1.21459967 1.57492004 1.30070419 1.80427247
   1.68929017 1.3232186 ]
  [1.66075959 1.14041294 1.21459967 1.57492004 1.30070419 1.80427247
   1.68929017 1.3232186 ]]

 [[1.61372929 1.22248829 1.25690953 1.72195651 1.54451353 1.23886972
   1.80321828 1.20342637]
  [1.61372929 1.22248829 1.25690953 1.72195651 1.54451353 1.23886972
   1.80321828 1.20342637]
  [1.61372929 1.22248829 1.25690953 1.72195651 1.54451353 1.23886972
   1.80321828 1.20342637]
  [1.61372929 1.22248829 1.25690953 1.72195651 1.54451353 1.23886972
   1.80321828 1.20342637]
  [1.61372929 1.22248829 1.25690953 1.72195651 1.54451353 1.23886972
   1.80321828 1.20342637]
  [1.61372929 1.22248829 1.25690953 1.72195651 1.54451353 1.23886972
   1.80321828 1.20342637]]]
```

The dimensions of x(6,8) and y(10,1,8) are different. However, it is possible to add them. Why is that? Also, change y(10,2,8) or y(10,1,4) and it will give ValueError. Can you find out why? (Hint: check rule 1).

For seeing NumPy mathematics at work, we will use the following example:

```
# Basic operations (+, -, *, /, %)
x = np.array([[1, 2, 3], [2, 3, 4]])
y = np.array([[1, 4, 9], [2, 3, -2]])
 # Add two array
add = np.add(x, y)
print(add)
# Subtract two array
sub = np.subtract(x, y)
print(sub)
# Multiply two array
mul = np.multiply(x, y)
print(mul) # Divide x, y
div = np.divide(x,y)
print(div)
# Calculated the remainder of x and y
rem = np.remainder(x, y)
print(rem)
```

```
[[ 2  6 12]
 [ 4  6  2]]
[[ 0 -2 -6]
 [ 0  0  6]]
[[ 1  8 27]
 [ 4  9 -8]]
[[ 1.          0.5         0.33333333]
 [ 1.          1.         -2.        ]]
[[0 2 3]
 [0 0 0]]
```

Let's now see how we can create a subset and slice an array using an index:

```
x = np.array([10, 20, 30, 40, 50])
# Select items at index 0 and 1
print(x[0:2])
 # Select item at row 0 and 1 and column 1 from 2D array
y = np.array([[ 1, 2, 3, 4], [ 9, 10, 11 ,12]])
print(y[0:2, 1])
 # Specifying conditions
biggerThan2 = (y >= 2)
print(y[biggerThan2])
```

```
[10 20]
[ 2 10]
[ 2  3  4  9 10 11 12]
```

Pandas