What is NumPy ?

- NumPy stands for Numerical Python
- It is a fundamental package for scientific computing in Python
- NumPy provides Python with an extensive math library capable of performing numerical computations effectively and efficiently

Why use NumPy ?

- Speed
- Multidimensional array data structures represents matricies and vectors that allow linear algebra operations efficiently
- Optimized built-in mathematical functions makes programs more readable and easier to understand

In [126…
```
import time
import numpy as np
x = np.random.random(100000000)

# Case 1
start = time.time()
sum(x) / len(x)
print(time.time() - start)

# Case 2
start = time.time()
np.mean(x)
print(time.time() - start)
```

```
4.396330118179321
0.01758408546447754
```

1. 1-D Array of Integers

In [127…
```
x = np.array([1, 2, 3, 4, 5])
print('x = ', x)
```

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

In [128…
```
print('x is an object of type:', type(x))
print('The elements in x are of type:', x.dtype)
print('x has dimensions:', x.shape)
print('x has size:', x.size)
```

```
x is an object of type: <class 'numpy.ndarray'>
The elements in x are of type: int64
x has dimensions: (5,)
x has size: 5
```

2. 1-D Array of Strings

In [129…
```
x = np.array(['Hello', 'World'])
print('x = ', x)
print('x has dimensions:', x.shape)
print('x is an object of type:', type(x))
print('The elements in x are of type:', x.dtype)
```

```
x =  ['Hello' 'World']
x has dimensions: (2,)
x is an object of type: <class 'numpy.ndarray'>
The elements in x are of type: <U5
```

3. different data types

In [130…
```
x = np.array([1, 2, 'World'])

print('x = ', x)
print('x has dimensions:', x.shape)
print('x is an object of type:', type(x))
print('The elements in x are of type:', x.dtype)
```

```
x =  ['1' '2' 'World']
x has dimensions: (3,)
x is an object of type: <class 'numpy.ndarray'>
The elements in x are of type: <U21
```

4. upcasting

In [131…
```
x = np.array([1,2,3])
y = np.array([1.0,2.0,3.0])
z = np.array([1, 2.5, 4])

print('The elements in x are of type:', x.dtype)
print('The elements in y are of type:', y.dtype)
print('The elements in z are of type:', z.dtype)
```

```
The elements in x are of type: int64
The elements in y are of type: float64
The elements in z are of type: float64
```

5. loss of precision

In [132…
```
# We create a rank 1 ndarray of floats but set the dtype to int64
x = np.array([1.5, 2.2, 3.7, 4.0, 5.9], dtype = np.int64)

# We print the dtype x
print('x = ', x)
print('The elements in x are of type:', x.dtype)
```

```
x =  [1 2 3 4 5]
The elements in x are of type: int64
```

6. 2-D Array

In [133…
```
Y = np.array([[1,2,3],[4,5,6],[7,8,9], [10,11,12]])

print('Y = \n', Y)
print('Y is an object of type:', type(Y))
print('The elements in Y are of type:', Y.dtype)
print('Y has dimensions:', Y.shape)
print('Y has a total of', Y.size, 'elements')
```

```
Y =
 [[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]
Y is an object of type: <class 'numpy.ndarray'>
The elements in Y are of type: int64
Y has dimensions: (4, 3)
Y has a total of 12 elements
```

7. Saving a Numpy Array

In [134…
```
# We create a rank 1 ndarray
x = np.array([1, 2, 3, 4, 5])

# We save x into the current directory as
np.save('my_array', x)
```

In [135…
```
# We load the saved array from our current directory into variable y
y = np.load('my_array.npy')

# We print information about the ndarray we loaded
print('Y = \n', y)
print('y is an object of type:', type(y))
print('The elements in y are of type:', y.dtype)
```

```
Y =
 [1 2 3 4 5]
y is an object of type: <class 'numpy.ndarray'>
The elements in y are of type: int64
```

8. Create a Numpy array of zeros with a desired shape

In [136…
```
# We create a 3 x 4 ndarray full of zeros.
X = np.zeros((3,4), dtype=int)

# We print X
print()
print('X = \n', X)
print()

# We print information about X
print('X has dimensions:', X.shape)
```

```
print('X is an object of type:', type(X))
print('The elements in X are of type:', X.dtype)
```

```
X =
 [[0 0 0 0]
 [0 0 0 0]
 [0 0 0 0]]

X has dimensions: (3, 4)
X is an object of type: <class 'numpy.ndarray'>
The elements in X are of type: int64
```

9. Create a Numpy array of ones

In [137…

```
# We create a 3 x 2 ndarray full of ones.
X = np.ones((3,2))

# We print X
print()
print('X = \n', X)
print()

# We print information about X
print('X has dimensions:', X.shape)
print('X is an object of type:', type(X))
print('The elements in X are of type:', X.dtype)
```

```
X =
 [[1. 1.]
 [1. 1.]
 [1. 1.]]

X has dimensions: (3, 2)
X is an object of type: <class 'numpy.ndarray'>
The elements in X are of type: float64
```

10. Create a Numpy array of constants

In [138…

```
# We create a 2 x 3 ndarray full of fives.
X = np.full((2,3), 5)

# We print X
print()
print('X = \n', X)
print()

# We print information about X
print('X has dimensions:', X.shape)
print('X is an object of type:', type(X))
print('The elements in X are of type:', X.dtype)
```

```
X =
 [[5 5 5]
 [5 5 5]]

X has dimensions: (2, 3)
X is an object of type: <class 'numpy.ndarray'>
The elements in X are of type: int64
```

11. Create a Numpy array of Identity matrix

In [139…

```
# We create a 5 x 5 Identity matrix.
X = np.eye(5)

# We print X
print()
print('X = \n', X)
print()

# We print information about X
print('X has dimensions:', X.shape)
print('X is an object of type:', type(X))
print('The elements in X are of type:', X.dtype)
```

```
X =
 [[1. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0.]
 [0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 1.]]

X has dimensions: (5, 5)
X is an object of type: <class 'numpy.ndarray'>
The elements in X are of type: float64
```

12. Create a Numpy array of constants

In [140…

```
# Create a 4 x 4 diagonal matrix that contains the numbers 10,20,30, and 50
# on its main diagonal
X = np.diag([10,20,30,50])

# We print X
print()
print('X = \n', X)
print()
```

```
X =
 [[10  0  0  0]
 [ 0 20  0  0]
 [ 0  0 30  0]
 [ 0  0  0 50]]
```

13. Create a Numpy array of evenly spaced values in a given range, using `arange(stop_val)`

See the complete syntax of `NumPy.arange()` here

In [141…
```
# We create a rank 1 ndarray that has sequential integers from 0 to 9
x = np.arange(10)

# We print the ndarray
print()
print('x = ', x)
print()

# We print information about the ndarray
print('x has dimensions:', x.shape)
print('x is an object of type:', type(x))
print('The elements in x are of type:', x.dtype)
```

```
x =  [0 1 2 3 4 5 6 7 8 9]

x has dimensions: (10,)
x is an object of type: <class 'numpy.ndarray'>
The elements in x are of type: int64
```

14. Create a Numpy array using `arange(start_val, stop_val)`

In [142…
```
# We create a rank 1 ndarray that has sequential integers from 4 to 9.
x = np.arange(4,10)

# We print the ndarray
print()
print('x = ', x)
print()

# We print information about the ndarray
print('x has dimensions:', x.shape)
print('x is an object of type:', type(x))
print('The elements in x are of type:', x.dtype)
```

```
x =  [4 5 6 7 8 9]

x has dimensions: (6,)
x is an object of type: <class 'numpy.ndarray'>
The elements in x are of type: int64
```

15. Create a Numpy array using `arange(start_val, stop_val, step_size)`

In [143…
```
# We create a rank 1 ndarray that has evenly spaced integers from 1 to 13 in
x = np.arange(1,14,3)

# We print the ndarray
print()
print('x = ', x)
```

```
print()

# We print information about the ndarray
print('x has dimensions:', x.shape)
print('x is an object of type:', type(x))
print('The elements in x are of type:', x.dtype)
```

```
x =  [ 1  4  7 10 13]

x has dimensions: (5,)
x is an object of type: <class 'numpy.ndarray'>
The elements in x are of type: int64
```

16. Create a Numpy array using `linspace(start, stop, n)`, with `stop` inclusive.

See the all possible arguments in the syntax here

In [144…
```
# We create a rank 1 ndarray that has 10 integers evenly spaced between 0 an
x = np.linspace(0,25,10)

# We print the ndarray
print()
print('x = \n', x)
print()

# We print information about the ndarray
print('x has dimensions:', x.shape)
print('x is an object of type:', type(x))
print('The elements in x are of type:', x.dtype)
```

```
x =
 [ 0.          2.77777778  5.55555556  8.33333333 11.11111111 13.88888889
 16.66666667 19.44444444 22.22222222 25.        ]

x has dimensions: (10,)
x is an object of type: <class 'numpy.ndarray'>
The elements in x are of type: float64
```

17. Create a Numpy array using `linspace(start, stop, n)`, with `stop` excluded.

In [145…
```
# We create a rank 1 ndarray that has 10 integers evenly spaced between 0 an
# with 25 excluded.
x = np.linspace(0,25,10, endpoint = False)

# We print the ndarray
print()
print('x = ', x)
print()

# We print information about the ndarray
print('x has dimensions:', x.shape)
```

```
print('x is an object of type:', type(x))
print('The elements in x are of type:', x.dtype)
```

```
x =  [ 0.   2.5  5.   7.5 10.  12.5 15.  17.5 20.  22.5]

x has dimensions: (10,)
x is an object of type: <class 'numpy.ndarray'>
The elements in x are of type: float64
```

18. Create a Numpy array by feeding the output of `arange()` function to the `reshape()` function.

Here, `arange()` function will give you a 1-D array, whereas the `reshape()` function will convert that 1-D array into a desired shape. **Remember, that the `size` of the final output must be as same as the `size` of the initial 1-D array.

In [146…

```
# We create a rank 1 ndarray with sequential integers from 0 to 19
x = np.arange(20)

# We print x
print()
print('Original x = ', x)
print()

# We reshape x into a 4 x 5 ndarray
x = np.reshape(x, (4,5))

# We print the reshaped x
print()
print('Reshaped x = \n', x)
print()

# We print information about the reshaped x
print('x has dimensions:', x.shape)
print('x is an object of type:', type(x))
print('The elements in x are of type:', x.dtype)
```

```
Original x =  [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19]


Reshaped x =
 [[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]]

x has dimensions: (4, 5)
x is an object of type: <class 'numpy.ndarray'>
The elements in x are of type: int64
```

19. Create a Numpy array by calling the `reshape()` function from the output of `arange()` function.

Notice the change in the arguments of `reshape()`

In [147…
```
# We create a a rank 1 ndarray with sequential integers from 0 to 19 and
# reshape it to a 4 x 5 array
Y = np.arange(20).reshape(4, 5)

# We print Y
print()
print('Y = \n', Y)
print()

# We print information about Y
print('Y has dimensions:', Y.shape)
print('Y is an object of type:', type(Y))
print('The elements in Y are of type:', Y.dtype)
```

```
Y =
 [[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]]

Y has dimensions: (4, 5)
Y is an object of type: <class 'numpy.ndarray'>
The elements in Y are of type: int64
```

20. Create a rank 2 Numpy array by using the `reshape()` function.

In [148…
```
# We create a rank 1 ndarray with 10 integers evenly spaced between 0 and 50
# with 50 excluded. We then reshape it to a 5 x 2 ndarray
X = np.linspace(0,50,10, endpoint=False).reshape(5,2)

# We print X
print()
print('X = \n', X)
print()

# We print information about X
print('X has dimensions:', X.shape)
print('X is an object of type:', type(X))
print('The elements in X are of type:', X.dtype)
```

```
X =
 [[ 0.  5.]
 [10. 15.]
 [20. 25.]
 [30. 35.]
 [40. 45.]]

X has dimensions: (5, 2)
X is an object of type: <class 'numpy.ndarray'>
The elements in X are of type: float64
```

21. Create a Numpy array using the `numpy.random.random()` function.

In [149…
```
# We create a 3 x 3 ndarray with random floats in the half-open interval [0.
X = np.random.random((3,3))

# We print X
print()
print('X = \n', X)
print()

# We print information about X
print('X has dimensions:', X.shape)
print('X is an object of type:', type(X))
print('The elements in x are of type:', X.dtype)
```

```
X =
 [[0.58017079 0.68231756 0.33301426]
 [0.65647285 0.52317526 0.9031139 ]
 [0.55373112 0.0163097  0.94402392]]

X has dimensions: (3, 3)
X is an object of type: <class 'numpy.ndarray'>
The elements in x are of type: float64
```

22. Create a Numpy array using the `numpy.random.randint()` function.

In [150…
```
# We create a 3 x 2 ndarray with random integers in the half-open interval [
X = np.random.randint(4,15,size=(3,2))

# We print X
print()
print('X = \n', X)
print()

# We print information about X
print('X has dimensions:', X.shape)
print('X is an object of type:', type(X))
print('The elements in X are of type:', X.dtype)
```

```
X =
 [[ 8 14]
 [ 6  7]
 [ 4  4]]

X has dimensions: (3, 2)
X is an object of type: <class 'numpy.ndarray'>
The elements in X are of type: int64
```

23. Create a Numpy array of "Normal" distributed random numbers, using the `numpy.random.normal()` function.

In [151…
```
# We create a 1000 x 1000 ndarray of random floats drawn from normal (Gaussi
# with a mean of zero and a standard deviation of 0.1.
X = np.random.normal(0, 0.1, size=(1000,1000))

# We print X
print()
print('X = \n', X)
print()

# We print information about X
print('X has dimensions:', X.shape)
print('X is an object of type:', type(X))
print('The elements in X are of type:', X.dtype)
print('The elements in X have a mean of:', X.mean())
print('The maximum value in X is:', X.max())
print('The minimum value in X is:', X.min())
print('X has', (X < 0).sum(), 'negative numbers')
print('X has', (X > 0).sum(), 'positive numbers')
print('std: ', X.std())
```

```
X =
 [[-0.04497681 -0.08509899 -0.08134305 ... -0.13212317  0.18522451
  -0.09952249]
 [-0.06585028 -0.2411061   0.03407872 ...  0.0477566  -0.08336192
  -0.14888973]
 [ 0.12175629  0.07498487 -0.0439271  ... -0.00773693 -0.03157929
  -0.09868149]
 ...
 [ 0.15927599 -0.09902532  0.13562035 ... -0.00603086  0.04901049
   0.01012967]
 [-0.18092174  0.11039198  0.03440118 ...  0.06352502  0.11978952
   0.00275207]
 [ 0.10055526 -0.04055822 -0.0275883  ... -0.05759455  0.04216732
   0.05198288]]

X has dimensions: (1000, 1000)
X is an object of type: <class 'numpy.ndarray'>
The elements in X are of type: float64
The elements in X have a mean of: 1.3632558437354121e-05
The maximum value in X is: 0.4685118210362107
The minimum value in X is: -0.47028897614371873
X has 500100 negative numbers
X has 499900 positive numbers
std:  0.09999530707074065
```

24. Access individual elements of 1-D array

In [152…
```
# We create a rank 1 ndarray that contains integers from 1 to 5
x = np.array([1, 2, 3, 4, 5])

# We print x
print()
print('x = ', x)
print()
```

```
# Let's access some elements with positive indices
print('This is First Element in x:', x[0])
print('This is Second Element in x:', x[1])
print('This is Fifth (Last) Element in x:', x[4])
print()

# Let's access the same elements with negative indices
print('This is First Element in x:', x[-5])
print('This is Second Element in x:', x[-4])
print('This is Fifth (Last) Element in x:', x[-1])
```

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

This is First Element in x: 1
This is Second Element in x: 2
This is Fifth (Last) Element in x: 5

This is First Element in x: 1
This is Second Element in x: 2
This is Fifth (Last) Element in x: 5
```

25. Modify an element of 1-D array

In [153…

```
# We create a rank 1 ndarray that contains integers from 1 to 5
x = np.array([1, 2, 3, 4, 5])

# We print the original x
print()
print('Original:\n x = ', x)
print()

# We change the fourth element in x from 4 to 20
x[3] = 20

# We print x after it was modified
print('Modified:\n x = ', x)
```

```
Original:
 x =  [1 2 3 4 5]

Modified:
 x =  [ 1  2  3 20  5]
```

26. Access individual elements of 2-D array

In [154…

```
# We create a 3 x 3 rank 2 ndarray that contains integers from 1 to 9
X = np.array([[1,2,3],[4,5,6],[7,8,9]])

# We print X
print()
print('X = \n', X)
print()

# Let's access some elements in X
print('This is (0,0) Element in X:', X[0,0])
```

```
print('This is (0,1) Element in X:', X[0,1])
print('This is (2,2) Element in X:', X[2,2])
```

```
X =
 [[1 2 3]
 [4 5 6]
 [7 8 9]]

This is (0,0) Element in X: 1
This is (0,1) Element in X: 2
This is (2,2) Element in X: 9
```

27. Modify an element of 2-D array

In [155…

```
# We create a 3 x 3 rank 2 ndarray that contains integers from 1 to 9
X = np.array([[1,2,3],[4,5,6],[7,8,9]])

# We print the original x
print()
print('Original:\n X = \n', X)
print()

# We change the (0,0) element in X from 1 to 20
X[0,0] = 20

# We print X after it was modified
print('Modified:\n X = \n', X)
```

```
Original:
 X =
 [[1 2 3]
 [4 5 6]
 [7 8 9]]

Modified:
 X =
 [[20  2  3]
 [ 4  5  6]
 [ 7  8  9]]
```

28. Delete elements

In [156…

```
# We create a rank 1 ndarray
x = np.array([1, 2, 3, 4, 5])

# We create a rank 2 ndarray
Y = np.array([[1,2,3],[4,5,6],[7,8,9]])

# We print x
print()
print('Original x = ', x)

# We delete the first and last element of x
x = np.delete(x, [0,4])
```

```
# We print x with the first and last element deleted
print()
print('Modified x = ', x)

# We print Y
print()
print('Original Y = \n', Y)

# We delete the first row of y
w = np.delete(Y, 0, axis=0)

# We delete the first and last column of y
v = np.delete(Y, [0,2], axis=1)

# We print w
print()
print('w = \n', w)

# We print v
print()
print('v = \n', v)
```

```
Original x =  [1 2 3 4 5]

Modified x =  [2 3 4]

Original Y =
 [[1 2 3]
 [4 5 6]
 [7 8 9]]

w =
 [[4 5 6]
 [7 8 9]]

v =
 [[2]
 [5]
 [8]]
```

29. Append elements

In [157…

```
# We create a rank 1 ndarray
x = np.array([1, 2, 3, 4, 5])

# We create a rank 2 ndarray
Y = np.array([[1,2,3],[4,5,6]])

# We print x
print()
print('Original x = ', x)

# We append the integer 6 to x
x = np.append(x, 6)
```

```
# We print x
print()
print('x = ', x)

# We append the integer 7 and 8 to x
x = np.append(x, [7,8])

# We print x
print()
print('x = ', x)

# We print Y
print()
print('Original Y = \n', Y)

# We append a new row containing 7,8,9 to y
v = np.append(Y, [[7,8,9]], axis=0)

# We append a new column containing 9 and 10 to y
q = np.append(Y,[[9],[10]], axis=1)

# We print v
print()
print('v = \n', v)

# We print q
print()
print('q = \n', q)
```

```
Original x =  [1 2 3 4 5]

x =  [1 2 3 4 5 6]

x =  [1 2 3 4 5 6 7 8]

Original Y =
 [[1 2 3]
 [4 5 6]]

v =
 [[1 2 3]
 [4 5 6]
 [7 8 9]]

q =
 [[ 1  2  3  9]
 [ 4  5  6 10]]
```

30. Insert elements

In [158…

```
# We create a rank 1 ndarray
x = np.array([1, 2, 5, 6, 7])

# We create a rank 2 ndarray
Y = np.array([[1,2,3],[7,8,9]])
```

```
# We print x
print()
print('Original x = ', x)

# We insert the integer 3 and 4 between 2 and 5 in x.
x = np.insert(x,2,[3,4])

# We print x with the inserted elements
print()
print('x = ', x)

# We print Y
print()
print('Original Y = \n', Y)

# We insert a row between the first and last row of y
w = np.insert(Y,1,[4,5,6],axis=0)

# We insert a column full of 5s between the first and second column of y
v = np.insert(Y,1,5, axis=1)

# We print w
print()
print('w = \n', w)

# We print v
print()
print('v = \n', v)
```

```
Original x =  [1 2 5 6 7]

x =  [1 2 3 4 5 6 7]

Original Y =
 [[1 2 3]
 [7 8 9]]

w =
 [[1 2 3]
 [4 5 6]
 [7 8 9]]

v =
 [[1 5 2 3]
 [7 5 8 9]]
```

31. Stack arrays

In [159…

```
# We create a rank 1 ndarray
x = np.array([1,2])

# We create a rank 2 ndarray
Y = np.array([[3,4],[5,6]])
```

```
# We print x
print()
print('x = ', x)

# We print Y
print()
print('Y = \n', Y)

# We stack x on top of Y
z = np.vstack((x,Y))

# We stack x on the right of Y. We need to reshape x in order to stack it on
w = np.hstack((Y,x.reshape(2,1)))

# We print z
print()
print('z = \n', z)

# We print w
print()
print('w = \n', w)
```

```
x =  [1 2]

Y =
 [[3 4]
 [5 6]]

z =
 [[1 2]
 [3 4]
 [5 6]]

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

32. Slicing in a 2-D ndarray

In [160…

```
# We create a 4 x 5 ndarray that contains integers from 0 to 19
X = np.arange(20).reshape(4, 5)

# We print X
print()
print('X = \n', X)
print()
```

```
X =
 [[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]]
```

In [161…

```
# We select all the elements that are in the 2nd through 4th rows and in the
# (The 1st being index 0)
```

```
Z = X[1:4,2:5]

# We print Z
print('Z = \n', Z)
```

```
Z =
 [[ 7  8  9]
 [12 13 14]
 [17 18 19]]
```

In [162…

```
# We can select the same elements as above using method 2
W = X[1:,2:5]

# We print W
print()
print('W = \n', W)
```

```
W =
 [[ 7  8  9]
 [12 13 14]
 [17 18 19]]
```

In [163…

```
# We select all the elements that are in the 1st through 3rd rows and in the
Y = X[:3,2:5]

# We print Y
print()
print('Y = \n', Y)

# We select all the elements in the 3rd row
v = X[2,:]

# We print v
print()
print('v = ', v)

# We select all the elements in the 3rd column
q = X[:,2]

# We print q
print()
print('q = ', q)

# We select all the elements in the 3rd column but return a rank 2 ndarray
R = X[:,2:3]

# We print R
print()
print('R = \n', R)
```

```
Y =
 [[ 2  3  4]
 [ 7  8  9]
 [12 13 14]]

v =  [10 11 12 13 14]

q =  [ 2  7 12 17]

R =
 [[ 2]
 [ 7]
 [12]
 [17]]
```

33. Slicing and editing elements in a 2-D ndarray

In [164…

```
# We create a 4 x 5 ndarray that contains integers from 0 to 19
X = np.arange(20).reshape(4, 5)

# We print X
print()
print('X = \n', X)
print()

# We select all the elements that are in the 2nd through 4th rows and in the
Z = X[1:4,2:5]

# We print Z
print()
print('Z = \n', Z)
print()

# We change the last element in Z to 555
Z[2,2] = 555

# We print X
print()
print('X = \n', X)
print()
```

```
X =
 [[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]]


Z =
 [[ 7  8  9]
 [12 13 14]
 [17 18 19]]


X =
 [[  0   1   2   3   4]
 [  5   6   7   8   9]
 [ 10  11  12  13  14]
 [ 15  16  17  18 555]]
```

34. Demonstrate the copy() function

In [165…

```
# We create a 4 x 5 ndarray that contains integers from 0 to 19
X = np.arange(20).reshape(4, 5)

# We print X
print()
print('X = \n', X)
print()

# create a copy of the slice using the np.copy() function
Z = np.copy(X[1:4,2:5])

#  create a copy of the slice using the copy as a method
W = X[1:4,2:5].copy()

# We change the last element in Z to 555
Z[2,2] = 555

# We change the last element in W to 444
W[2,2] = 444

# We print X
print()
print('X = \n', X)

# We print Z
print()
print('Z = \n', Z)

# We print W
print()
print('W = \n', W)
```

```
X =
 [[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]]


X =
 [[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]]

Z =
 [[  7   8   9]
 [ 12  13  14]
 [ 17  18 555]]

W =
 [[  7   8   9]
 [ 12  13  14]
 [ 17  18 444]]
```

35. Use an array as indices to either make slices, select, or change elements

In [166…

```
# We create a 4 x 5 ndarray that contains integers from 0 to 19
X = np.arange(20).reshape(4, 5)

# We create a rank 1 ndarray that will serve as indices to select elements f
indices = np.array([1,3])

# We print X
print()
print('X = \n', X)
print()

# We print indices
print('indices = ', indices)
print()

# We use the indices ndarray to select the 2nd and 4th row of X
Y = X[indices,:]

# We use the indices ndarray to select the 2nd and 4th column of X
Z = X[:, indices]

# We print Y
print()
print('Y = \n', Y)

# We print Z
print()
print('Z = \n', Z)
```

```
X =
 [[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]]

indices =  [1 3]


Y =
 [[ 5  6  7  8  9]
 [15 16 17 18 19]]

Z =
 [[ 1  3]
 [ 6  8]
 [11 13]
 [16 18]]
```

36. Use an array as indices to extract specific rows from a rank 2 ndarray.

In [167…

```
import numpy as np
# Let's create a rank 2 ndarray
X = np.random.randint(1,20, size=(50,5))
print("Shape of X is: ", X.shape)

# Create a rank 1 ndarray that contains a randomly chosen 10 values between
# The row_indices would represent the indices of rows of X
row_indices = np.random.randint(0,50, size=10)
print("Random 10 indices are: ", row_indices)

# To Do 1 - Print those rows of X whose indices are represented by entire ro
# Hint - Use the row_indices ndarray to select specified rows of X
X_subset = X[row_indices, :]
print(X_subset)
print()

# To Do 2 - Print those rows of X whose indices are present in row_indices[4
X_subset = X[row_indices[4:8], :]
print(X_subset)
```

```
Shape of X is:  (50, 5)
Random 10 indices are:  [ 0 41  2 48 26 24 19 32 12  6]
[[19  8  2  5 10]
 [18 16 19 15  4]
 [ 1  3 18 19  4]
 [11  1 18  5  5]
 [19 10  4  6 17]
 [ 9  6  9 15 18]
 [17 13 13 19  1]
 [ 2  2  9 18 13]
 [ 6  3 15 13 13]
 [19 15  9 11  6]]

[[19 10  4  6 17]
 [ 9  6  9 15 18]
 [17 13 13 19  1]
 [ 2  2  9 18 13]]
```

37. Demonstrate the diag() function

In [168…

```
# We create a 5 x 5 ndarray that contains integers from 0 to 24
X = np.arange(25).reshape(5, 5)

# We print X
print()
print('X = \n', X)
print()

# We print the elements in the main diagonal of X
print('z =', np.diag(X))
print()

# We print the elements above the main diagonal of X
print('y =', np.diag(X, k=1))
print()

# We print the elements below the main diagonal of X
print('w = ', np.diag(X, k=-1))
```

```
X =
 [[ 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]]

z = [ 0  6 12 18 24]

y = [ 1  7 13 19]

w =  [ 5 11 17 23]
```

38. Demonstrate the unique() function

In [169…
```
# Create 3 x 3 ndarray with repeated values
X = np.array([[1,2,3],[5,2,8],[1,2,3]])

# We print X
print()
print('X = \n', X)
print()

# We print the unique elements of X
print('The unique elements in X are:',np.unique(X))
```

```
X =
 [[1 2 3]
 [5 2 8]
 [1 2 3]]

The unique elements in X are: [1 2 3 5 8]
```

39. Boolean indexing

In [170…
```
# We create a 5 x 5 ndarray that contains integers from 0 to 24
X = np.arange(25).reshape(5, 5)

# We print X
print()
print('Original X = \n', X)
print()

# We use Boolean indexing to select elements in X:
print('The elements in X that are greater than 10:', X[X > 10])
print('The elements in X that less than or equal to 7:', X[X <= 7])
print('The elements in X that are between 10 and 17:', X[(X > 10) & (X < 17)

# We use Boolean indexing to assign the elements that are between 10 and 17
X[(X > 10) & (X < 17)] = -1

# We print X
print()
print('X = \n', X)
print()
```

```
Original X =
 [[ 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]]

The elements in X that are greater than 10: [11 12 13 14 15 16 17 18 19 20 2
1 22 23 24]
The elements in X that less than or equal to 7: [0 1 2 3 4 5 6 7]
The elements in X that are between 10 and 17: [11 12 13 14 15 16]

X =
 [[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 -1 -1 -1 -1]
 [-1 -1 17 18 19]
 [20 21 22 23 24]]
```

40. Set operations

In [171…

```
# We create a rank 1 ndarray
x = np.array([1,2,3,4,5])

# We create a rank 1 ndarray
y = np.array([6,7,2,8,4])

# We print x
print()
print('x = ', x)

# We print y
print()
print('y = ', y)

# We use set operations to compare x and y:
print()
print('The elements that are both in x and y:', np.intersect1d(x,y))
print('The elements that are in x that are not in y:', np.setdiff1d(x,y))
print('All the elements of x and y:',np.union1d(x,y))
```

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

y =  [6 7 2 8 4]

The elements that are both in x and y: [2 4]
The elements that are in x that are not in y: [1 3 5]
All the elements of x and y: [1 2 3 4 5 6 7 8]
```

41. Sort arrays using sort() function

In [172…

```
# We create an unsorted rank 1 ndarray
x = np.random.randint(1,11,size=(10,))
```

```
# We print x
print()
print('Original x = ', x)

# We sort x and print the sorted array using sort as a function.
print()
print('Sorted x (out of place):', np.sort(x))

# When we sort out of place the original array remains intact. To see this w
print()
print('x after sorting:', x)
```

```
Original x =  [10  8  7 10  9  5  9  5 10  8]

Sorted x (out of place): [ 5  5  7  8  8  9  9 10 10 10]

x after sorting: [10  8  7 10  9  5  9  5 10  8]
```

42. Sort rank-1 arrays using sort() method

In [173…

```
# We create an unsorted rank 1 ndarray
x = np.random.randint(1,11,size=(10,))

# We print x
print()
print('Original x = ', x)

# We sort x and print the sorted array using sort as a method.
x.sort()

# When we sort in place the original array is changed to the sorted array. T
print()
print('x after sorting:', x)
```

```
Original x =  [ 2  1  5 10  7 10  5 10  7  8]

x after sorting: [ 1  2  5  5  7  7  8 10 10 10]
```

43. Sort rank-2 arrays by specific axis

In [174…

```
# We create an unsorted rank 2 ndarray
X = np.random.randint(1,11,size=(5,5))

# We print X
print()
print('Original X = \n', X)
print()

# We sort the columns of X and print the sorted array
print()
print('X with sorted columns :\n', np.sort(X, axis = 0))

# We sort the rows of X and print the sorted array
print()
print('X with sorted rows :\n', np.sort(X, axis = 1))
```

```
Original X =
 [[ 4  5  9  4  2]
 [ 3 10  5  6  5]
 [ 6  1 10  7  1]
 [ 7  1  5  4  3]
 [ 5  9  7  2 10]]


X with sorted columns :
 [[ 3  1  5  2  1]
 [ 4  1  5  4  2]
 [ 5  5  7  4  3]
 [ 6  9  9  6  5]
 [ 7 10 10  7 10]]

X with sorted rows :
 [[ 2  4  4  5  9]
 [ 3  5  5  6 10]
 [ 1  1  6  7 10]
 [ 1  3  4  5  7]
 [ 2  5  7  9 10]]
```

44. Element-wise arithmetic operations on 1-D arrays

In [175…

```
# We create two rank 1 ndarrays
x = np.array([1,2,3,4])
y = np.array([5.5,6.5,7.5,8.5])

# We print x
print()
print('x = ', x)

# We print y
print()
print('y = ', y)
print()

# We perfrom basic element-wise operations using arithmetic symbols and func
print('x + y = ', x + y)
print('add(x,y) = ', np.add(x,y))
print()
print('x - y = ', x - y)
print('subtract(x,y) = ', np.subtract(x,y))
print()
print('x  *y = ', x*  y)
print('multiply(x,y) = ', np.multiply(x,y))
print()
print('x / y = ', x / y)
print('divide(x,y) = ', np.divide(x,y))
```

```
x =  [1 2 3 4]

y =  [5.5 6.5 7.5 8.5]

x + y =  [ 6.5  8.5 10.5 12.5]
add(x,y) =  [ 6.5  8.5 10.5 12.5]

x - y =  [-4.5 -4.5 -4.5 -4.5]
subtract(x,y) =  [-4.5 -4.5 -4.5 -4.5]

x  *y =  [ 5.5 13.  22.5 34. ]
multiply(x,y) =  [ 5.5 13.  22.5 34. ]

x / y =  [0.18181818 0.30769231 0.4        0.47058824]
divide(x,y) =  [0.18181818 0.30769231 0.4        0.47058824]
```

45. Element-wise arithmetic operations on a 2-D array (Same shape)

In [176…

```
# We create two rank 2 ndarrays
X = np.array([1,2,3,4]).reshape(2,2)
Y = np.array([5.5,6.5,7.5,8.5]).reshape(2,2)

# We print X
print()
print('X = \n', X)

# We print Y
print()
print('Y = \n', Y)
print()

# We perform basic element-wise operations using arithmetic symbols and func
print('X + Y = \n', X + Y)
print()
print('add(X,Y) = \n', np.add(X,Y))
print()
print('X - Y = \n', X - Y)
print()
print('subtract(X,Y) = \n', np.subtract(X,Y))
print()
print('X  *Y = \n', X*  Y)
print()
print('multiply(X,Y) = \n', np.multiply(X,Y))
print()
print('X / Y = \n', X / Y)
print()
print('divide(X,Y) = \n', np.divide(X,Y))
```

```
X =
 [[1 2]
 [3 4]]

Y =
 [[5.5 6.5]
 [7.5 8.5]]

X + Y =
 [[ 6.5  8.5]
 [10.5 12.5]]

add(X,Y) =
 [[ 6.5  8.5]
 [10.5 12.5]]

X - Y =
 [[-4.5 -4.5]
 [-4.5 -4.5]]

subtract(X,Y) =
 [[-4.5 -4.5]
 [-4.5 -4.5]]

X  *Y =
 [[ 5.5 13. ]
 [22.5 34. ]]

multiply(X,Y) =
 [[ 5.5 13. ]
 [22.5 34. ]]

X / Y =
 [[0.18181818 0.30769231]
 [0.4        0.47058824]]

divide(X,Y) =
 [[0.18181818 0.30769231]
 [0.4        0.47058824]]
```

46. Additional mathematical functions

In [177…

```
# We create a rank 1 ndarray
x = np.array([1,2,3,4])

# We print x
print()
print('x = ', x)

# We apply different mathematical functions to all elements of x
print()
print('EXP(x) =', np.exp(x))
print()
print('SQRT(x) =',np.sqrt(x))
print()
```

```
print('POW(x,2) =',np.power(x,2)) # We raise all elements to the power of 2
```

```
x =  [1 2 3 4]

EXP(x) = [ 2.71828183  7.3890561  20.08553692 54.59815003]

SQRT(x) = [1.         1.41421356 1.73205081 2.        ]

POW(x,2) = [ 1  4  9 16]
```

47. Statistical functions

In [178…

```
# We create a 2 x 2 ndarray
X = np.array([[1,2], [3,4]])

# We print x
print()
print('X = \n', X)
print()

print('Average of all elements in X:', X.mean())
print('Average of all elements in the columns of X:', X.mean(axis=0))
print('Average of all elements in the rows of X:', X.mean(axis=1))
print()
print('Sum of all elements in X:', X.sum())
print('Sum of all elements in the columns of X:', X.sum(axis=0))
print('Sum of all elements in the rows of X:', X.sum(axis=1))
print()
print('Standard Deviation of all elements in X:', X.std())
print('Standard Deviation of all elements in the columns of X:', X.std(axis=
print('Standard Deviation of all elements in the rows of X:', X.std(axis=1))
print()
print('Median of all elements in X:', np.median(X))
print('Median of all elements in the columns of X:', np.median(X,axis=0))
print('Median of all elements in the rows of X:', np.median(X,axis=1))
print()
print('Maximum value of all elements in X:', X.max())
print('Maximum value of all elements in the columns of X:', X.max(axis=0))
print('Maximum value of all elements in the rows of X:', X.max(axis=1))
print()
print('Minimum value of all elements in X:', X.min())
print('Minimum value of all elements in the columns of X:', X.min(axis=0))
print('Minimum value of all elements in the rows of X:', X.min(axis=1))
```

```
X =
 [[1 2]
 [3 4]]

Average of all elements in X: 2.5
Average of all elements in the columns of X: [2. 3.]
Average of all elements in the rows of X: [1.5 3.5]

Sum of all elements in X: 10
Sum of all elements in the columns of X: [4 6]
Sum of all elements in the rows of X: [3 7]

Standard Deviation of all elements in X: 1.1180339887498895
Standard Deviation of all elements in the columns of X: [1. 1.]
Standard Deviation of all elements in the rows of X: [0.5 0.5]

Median of all elements in X: 2.5
Median of all elements in the columns of X: [2. 3.]
Median of all elements in the rows of X: [1.5 3.5]

Maximum value of all elements in X: 4
Maximum value of all elements in the columns of X: [3 4]
Maximum value of all elements in the rows of X: [2 4]

Minimum value of all elements in X: 1
Minimum value of all elements in the columns of X: [1 2]
Minimum value of all elements in the rows of X: [1 3]
```

48. Change value of all elements of an array

In [179…

```
# We create a 2 x 2 ndarray
X = np.array([[1,2], [3,4]])

# We print x
print()
print('X = \n', X)
print()

print('3  *X = \n', 3*  X)
print()
print('3 + X = \n', 3 + X)
print()
print('X - 3 = \n', X - 3)
print()
print('X / 3 = \n', X / 3)
```

```
X =
 [[1 2]
 [3 4]]

3  *X =
 [[ 3  6]
 [ 9 12]]

3 + X =
 [[4 5]
 [6 7]]

X - 3 =
 [[-2 -1]
 [ 0  1]]

X / 3 =
 [[0.33333333 0.66666667]
 [1.         1.33333333]]
```

49. Arithmetic operations on 2-D arrays (Compatible shape)

In [180…

```
# We create a rank 1 ndarray
x = np.array([1,2,3])

# We create a 3 x 3 ndarray
Y = np.array([[1,2,3],[4,5,6],[7,8,9]])

# We create a 3 x 1 ndarray
Z = np.array([1,2,3]).reshape(3,1)

# We print x
print()
print('x = ', x)
print()

# We print Y
print()
print('Y = \n', Y)
print()

# We print Z
print()
print('Z = \n', Z)
print()

print('x + Y = \n', x + Y)
print()
print('Z + Y = \n',Z + Y)
```

```
x =  [1 2 3]


Y =
 [[1 2 3]
 [4 5 6]
 [7 8 9]]


Z =
 [[1]
 [2]
 [3]]

x + Y =
 [[ 2  4  6]
 [ 5  7  9]
 [ 8 10 12]]

Z + Y =
 [[ 2  3  4]
 [ 6  7  8]
 [10 11 12]]
```

In [ ]: