NUMPY

 # easiest way to create an array is by using an array function

```
import numpy as np  # I am importing numpy as np

scores = [89,56.34, 76,89, 98]

first_arr =np.array(scores)

print(first_arr)

print(first_arr.dtype)  # .dtype return the data type of the array object
```

 # Nested lists with equal length, will be converted into a multidimensional array

```
scores_1 = [[34,56,23,89], [11,45,76,34]]

second_arr = np.array(scores_1)

print(second_arr)

print (second_arr.ndim)  #.ndim gives you the dimensions of an array.

print (second_arr.shape) #(number of rows, number of columns)

print (second_arr.dtype)
```

```
x = np.zeros(10) # returns a array of zeros, the same applies for np.ones(10)
print(x)
```

```
y=np.zeros((4,3)) # you can also mention the shape of the array
print(y)
```

```
np.arange(15)
```

```
np.eye(6) # Create a square N x N identity matrix (1's on the diagonal and 0's elsewhere)
```

#Batch operations on data can be performed without using for loops, this is called vectorization

scores = [89,56.34, 76,89, 98]

first_arr =np.array(scores)

print (first_arr)

print (first_arr * first_arr)

print (first_arr - first_arr)

print (1/(first_arr))

print (first_arr ** 0.5)

Indexing and Slicing

 # you may want to select a subset of your data, for which Numpy array indexing is really useful

new_arr = np.arange(12)

print (new_arr)

print (new_arr[4:9])

new_arr[4:9] = 99 **#assign sequence of values from 4 to 9 as 99**

print (new_arr)

 # A major diffence between lists and array is that, array slices are views on the original array. This means that the data is not copied, and any modifications to the view will be reflected in the source array.

modi_arr = new_arr[4:9]

modi_arr[1] = 123456

print (new_arr)   # you can see the changes are refelected in main array.

modi_arr[:]   # the sliced variable

 # arrays can be treated like matrices

matrix_arr =np.array([[3,4,5],[6,7,8],[9,5,1]])

print (matrix_arr)

print (matrix_arr[1])

print (matrix_arr[0][2]) **#first row and third column**

print (matrix_arr[0,2]) **# This is same as the above operation**

	Column 0	Column 1	Column 2
Row 0	0,0	0,1	0,2
Row 1	1,0	1,1	1,2
Row 2	2,0	2,1	2,2

 # 3d arrays -> this is a 2x2x3 array

three_d_arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])

print (three_d_arr)

print ("returns the second list inside first list {}".format(three_d_arr[0,1]))

three_d_arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])

print (three_d_arr[0])

#if you omit later indices, the returned object will be a lower dimensional ndarray consisting of all the data along the higher dimensions

copied_values = three_d_arr[0].copy() **# copy arr[0] value to copied_values** three_d_arr[0]= 99 **# change all values of arr[0] to 99**

print("New value of three_d_arr: {}".format(three_d_arr)) **# check the new value of three_d_arr**

three_d_arr[0] = copied_values **# assign copied values back to three_d_arr[0]**

print(" three_d_arr again: {}".format(three_d_arr))

matrix_arr =np.array([[3,4,5],[6,7,8],[9,5,1]])

print ("The original matrix {}:".format(matrix_arr))

print ("slices the first two rows:{}".format(matrix_arr[:2])) **# similar to list slicing. returns first two rows of the array**

print ("Slices the first two rows and two columns:{}".format(matrix_arr[:2, 1:]))

print ("returns 6 and 7: {}".format(matrix_arr[1,:2]))

print ("Returns first column: {}".format(matrix_arr[:,:1])) **#Note that a colon by itself means to take the entire axis**

personals = np.array(['Manu', 'Jeevan', 'Prakash', 'Manu', 'Prakash', 'Jeevan', 'Prakash'])

print(personals == 'Manu') **#checks for the string 'Manu' in personals. If present it returns true; else false#**

#Import random module from Numpy

from numpy import random

random_no = random.randn(7,4)

print (random_no)

random_no[personals =='Manu'] **#The function returns the rows for which the value of manu is true**

 # To select everything except 'Manu', you can use != or negate the condition using -:

print(personals != 'Manu')

random_no[(personals == 'Manu')] **#get everything except 1st and 4th rows**

```
# you can use boolean operator &(and), |(or)

new_variable = (personals == 'Manu') | (personals == 'Jeevan')

print(new_variable)

random_no[new_variable]


random_no[random_no < 0] =0

random_no  # This will set all negative values to zero


random_no[ personals != 'Manu'] = 9  # This will set all rows except 1 and 4 to 9.

random_no


from numpy import random
algebra = random.randn(7,4) # empty will return a matrix of size 7,4
for j in range(7):
   algebra[j] = j
algebra


# To select a subset of rows in particular order, you can simply pass a list.
algebra[[4,5,1]]   #returns a subset of rows


fancy = np.arange(36).reshape(9,4) #reshape is to reshape an array

print(fancy)

fancy[[1,4,3,2],[3,2,1,0]]  #the position of the output array are[(1,3),(4,2),(3,1),(2,0)]


fancy[[1, 4, 8, 2]][:, [0, 3, 1, 2]]  # entire first row is selected, but the elements are interchanged, same goes for 4th, 8th and 2 nd row.
```

 # another way to do the above operation is by using np.ix_ function.

fancy[np.ix_([1,4,8,2],[0,3,1,2])]

#Transposing Arrays

transpose= np.arange(12).reshape(3,4)

transpose.T **# the shape has changed to 4,3**

#you can use np.dot function to perform matrix computations. You can calculate X transpose X as follows:

np.dot(transpose.T, transpose)

#universal functions They perform element wise operations on data in arrays.

funky =np.arange(8)

print (np.sqrt(funky))

print (np.exp(funky)) **#exponent of the array**

 # these are called as unary functions

Function	Description
abs, fabs	Compute the absolute value element-wise for integer, floating point, or complex values. Use fabs as a faster alternative for non-complex-valued data
sqrt	Compute the square root of each element. Equivalent to arr ** 0.5
square	Compute the square of each element. Equivalent to arr ** 2
exp	Compute the exponent e^x of each element
log, log10, log2, log1p	Natural logarithm (base *e*), log base 10, log base 2, and log(1 + x), respectively
sign	Compute the sign of each element: 1 (positive), 0 (zero), or -1 (negative)
ceil	Compute the ceiling of each element, i.e. the smallest integer greater than or equal to each element
floor	Compute the floor of each element, i.e. the largest integer less than or equal to each element
rint	Round elements to the nearest integer, preserving the dtype
modf	Return fractional and integral parts of array as separate array
isnan	Return boolean array indicating whether each value is NaN (Not a Number)
isfinite, isinf	Return boolean array indicating whether each element is finite (non-inf, non-NaN) or infinite, respectively
cos, cosh, sin, sinh, tan, tanh	Regular and hyperbolic trigonometric functions
arccos, arccosh, arcsin, arcsinh, arctan, arctanh	Inverse trigonometric functions
logical_not	Compute truth value of not x element-wise. Equivalent to -arr.

 # Binary functions take two value, Others such as maximum, add

x = random.randn(10)

y = random.randn(10)

print(x)

print(y)

print(np.maximum(x,y)) **# element wise operation**

print(np.modf(x)) **# function modf returns the fractional and integral parts of a floating point arrays**

#List of binary functions available

#logical operators , and greater, greater_equal,less, less_equal, equal, not_equal operations can also be performed

Function	Description
`add`	Add corresponding elements in arrays
`subtract`	Subtract elements in second array from first array
`multiply`	Multiply array elements
`divide, floor_divide`	Divide or floor divide (truncating the remainder)
`power`	Raise elements in first array to powers indicated in second array
`maximum, fmax`	Element-wise maximum. `fmax` ignores NaN
`minimum, fmin`	Element-wise minimum. `fmin` ignores NaN
`mod`	Element-wise modulus (remainder of division)
`copysign`	Copy sign of values in second argument to values in first argument

#Data processing using Arrays

mtrices = np.arange(-5,5,1)

x, y = np.meshgrid(mtrices, mtrices) **#mesh grid function takes two 1 d arrays and produces two 2d arrays**

print("Matrix values of y: {}".format(y))

print("Matrix values of x: {}".format(x))

`zip()`

#When you `zip()` together three lists containing 20 elements each, the result has twenty elements. Each element is a three-tuple.

x1= np.array([1,2,3,4,5])

y1 = np.array([6,7,8,9,10])

cond =[True, False, True, True, False]

#If you want to take a value from x1 whenever the corresponding value in cond is true, otherwise take value from y.

z1 = [(x,y,z) for x,y,z in zip(x1, y1, cond)] **# I have used zip function To illustrate the concept**

print(z1)

np.where(cond, x1, y1)

ra = np.random.randn(5,5)

 # If you want to replace negative values in ra with -1 and positive values with 1. You can do it using where function

print(ra)

print(np.where(ra>0, 1, -1)) **# If values in ra are greater than zero, replace it with 1, else replace it with -1.**

 # to set only positive values

np.where(ra >0, 1, ra) **# same implies to negative values**

Statistical methods

thie = np.random.randn(5,5)

print(thie.mean()) **# calculates the mean of thie**

print(np.mean(thie)) **# alternate method to calculate mean**

print(thie.sum())

jp =np.arange(12).reshape(4,3)

print("The arrays are: ",jp)

print("The sum of rows are :",np.sum(jp, axis =0)) **#axis =0, gives you sum of the columns. axis =1 , gives sum of rows.**

 # remember this zero is for columns and one is for rows.

print(jp.sum(1)) **#returns sum of rows**

xp =np.random.randn(100)

print((xp > 0).sum()) **# sum of all positive values**

print((xp < 0).sum())

tandf =np.array([True,False,True,False,True,False])

print(tandf.any()) **#checks if any of the values are true**

print(tandf.all()) **#returns false even if a single value is false**

#These methods also work with non-boolean arrays, where non-zero elements evaluate to True.

Other array functions are:
std, var -> standard deviation and variance
min, max -> Minimum and Maximum
argmin, argmax -> Indices of minimum and maximum elements

#Sorting

lp = np.random.randn(8)

print(lp)

lp.sort()

lp

tp = np.random.randn(4,4)

tp

tp.sort(1)   **#check if the rows are sorted**

tp

personals = np.array(['Manu', 'Jeevan', 'Prakash', 'Manu', 'Prakash', 'Jeevan', 'Prakash'])

np.unique(personals) **# returns the unique elements in the array**

set(personals) **# set is an alternative to unique function**

np.in1d(personals, ['Manu'])   **#in1d function checks for the value 'Manu' and returns True, other wise returns False**

Other Functions are :
intersect1d(x, y)-> Compute the sorted, common elements in x and y
union1d(x,y) -> compute the sorted union of elements
setdiff1d(x,y) -> set difference, elements in x that are not in y
setxor1d(x, y) -> Set symmetric differences; elements that are in either of the arrays, but not both

cp = np.array([[1,2,3],[4,5,6]])

dp = np.array([[7,8],[9,10],[11,12]])

print("CP array : ",cp)

print("DP array : ",dp)

 # element wise multiplication

cp.dot(dp) # this is equivalent to np.dot(x,y) np.dot(cp, np.ones(3))

#Linear Algebra

 # numpy.linalg has standard matrix operations like determinants and inverse.

from numpy.linalg import inv, qr

cp = np.array([[1,2,3],[4,5,6]])

new_mat = cp.T.dot(cp) **# multiply cp inverse and cp, this is element wise multiplication**

print(new_mat)

print(cp)

print(cp.T)

sp = np.random.randn(5,5)

print(inv(sp))

rt = inv(sp)

 # to calculate the product of a matrix and its inverse

sp.dot(rt)

q,r = qr(sp)

print(q)

r

Other Matrix Functions

- diag : Return the diagonal (or off-diagonal) elements of a square matrix as a 1D array, or convert a 1D array into a square matrix with zeros on the off-diagonal
- trace: Compute the sum of the diagonal elements
- det: Compute the matrix determinant
- eig: Compute the eigenvalues and eigenvectors of a square matrix
- pinv: Compute the pseudo-inverse of a square matrix
- svd: Compute the singular value decomposition (SVD)
- solve: Solve the linear system Ax = b for x, where A is a square matrix
- lstsq: Compute the least-squares solution to y = Xb