NumPy Math Operations by Mrittika Megaraj

In [1]:
```
import numpy as np
```

Array Operations in NumPy

Arithmetic Operations:

In [2]:
```
arr1 = np.arange(9, dtype='i').reshape(3,3)
arr2 = np.arange(10,19, dtype='i').reshape(3,3)

print(arr1)
print(arr2)
```

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

In [3]:
```
print(np.add(arr1, arr2))
```

```
[[10 12 14]
 [16 18 20]
 [22 24 26]]
```

In [4]:
```
print(np.subtract(arr1, arr2))
```

```
[[-10 -10 -10]
 [-10 -10 -10]
 [-10 -10 -10]]
```

In [5]:
```
print(np.multiply(arr1, arr2))
```

```
[[  0  11  24]
 [ 39  56  75]
 [ 96 119 144]]
```

In [6]:
```
print(np.divide(arr1, arr2))
```

```
[[0.         0.09090909 0.16666667]
 [0.23076923 0.28571429 0.33333333]
 [0.375      0.41176471 0.44444444]]
```

numpy.reciprocal():

- This method returns the argument's element-by-element inverse.
- When an element's absolute value is greater than 1, the outcome is always 0, and an overflow warning is shown for integer 0.

In [7]:
```
ar = np.array([12, 13, 14, 15, 16, 17, 18, 19, 20]).reshape(3, 3)
print(np.reciprocal(ar))
```

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

In [8]:
```
ar = np.array([12, 13.7, 14, 15.2, 16, 17.6, 18.9, 19.5, 20]).reshape(3, 3)
print(np.reciprocal(ar))
```

```
[[0.08333333 0.0729927  0.07142857]
 [0.06578947 0.0625     0.05681818]
 [0.05291005 0.05128205 0.05      ]]
```

numpy.power():

- This function treats the original array's elements as the base in the exponents' syntax, which then raises them to the power of the adjacent elements provided in the second array argument.

In [9]:
```
arr = np.array([1,2,3,4,5,6,7,8,9]).reshape(3,3)
print(np.power(arr, 2))
print(np.power(arr, 3))
```

```
[[ 1  4  9]
 [16 25 36]
 [49 64 81]]
[[  1   8  27]
 [ 64 125 216]
 [343 512 729]]
```

numpy.mod():

- This function returns the remainder of the division of the corresponding elements in the input array.
- The function numpy.remainder() also produces the same result.

In [10]:
```
arr1 = np.array([5, 10, 15])
arr2 = np.array([2, 4, 3])

print(np.mod(arr1, arr2))
print(np.remainder(arr1, arr2))
```

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

numpy.dot():

- We will begin with the cases in which both arguments are scalars or one-dimensional arrays.

In [11]:
```
print(np.dot(10, 12))

a = np.array([12])
b = np.array([15])
print(np.dot(a,b))

a1 = np.array([11, 10])
a2 = np.array([2, 4])
print(np.dot(a1, a2))
```

```
120
180
62
```

Operations on Complex Numbers

numpy.real():

- This function will return the real part of the given complex argument.

In [12]:
```
a = np.array([-6.6j, 0.9j, 14. , 1+9j])

print('Our complex array is:')
print(a)

print('Applying the numpy real function: ')
print(np.real(a))
```

```
Our complex array is:
[-0.-6.6j  0.+0.9j 14.+0.j   1.+9.j ]
Applying the numpy real function:
[-0.  0. 14.  1.]
```

numpy.imag():

- This function will return the imaginary part of the complex argument.

In [13]:
```
print('Applying the numpy imag function: ')
print(np.imag(a))
```

```
Applying the numpy imag function:
[-6.6  0.9  0.   9. ]
```

numpy.conj():

- This function will return the complex conjugate of the given complex argument.
- It is obtained by swapping the sign of the imaginary part.

In [14]:
```
print('Applying the numpy conj function: ')
print(np.conj(a))
```

```
Applying the numpy conj function:
[-0.+6.6j  0.-0.9j 14.-0.j   1.-9.j ]
```

numpy.angle():

- This function will return the angle of the given complex argument.
- The function has a parameter having the keyword- degree.
- If set to true, the function will return the angle in degrees; otherwise, the angle is returned in radians.

In [15]:
```
print('Applying the numpy angle function: ')
print(np.angle(a))

print('Applying the numpy angle function again (result in degrees)')
print(np.angle(a, deg = True))
```

```
Applying the numpy angle function:
[-1.57079633  1.57079633  0.          1.46013911]
Applying the numpy angle function again (result in degrees)
[-90.          90.           0.          83.65980825]
```

Using Numpy Arrays with Conditional Expressions

In [16]:
```
arr = np.array([1,2,3,4,5,6,7,8,9])

print(arr[arr%2 == 0])
print(arr[arr%2 == 1])

odd = arr%2 == 1
even = arr%2 == 0

print(arr[odd])
print(arr[even])
```

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

Logical Operators

- The logical operators "or" and "and" also apply to numpy arrays elementwise.
- For this, we can use the numpy logical_or and logical_and methods.

In [17]:
```
arr1 = np.array([[True, False], [True, True]])
arr2 = np.array([[False, False], [False, True]])

print(np.logical_or(arr1, arr2))
print(np.logical_and(arr1, arr2))
```

```
[[ True False]
 [ True  True]]
[[False False]
 [False  True]]
```

Broadcasting:

- We take a larger dimension array and a smaller dimension array, and we convert or extend the smaller dimension array to the larger dimension array multiple times to carry out an operation.
- To put this in another way, the smaller array can occasionally be "broadcasted" so that it takes on the same dimension as the larger array.

In [18]:
```
arr1 = np.array([[1,2,3], [4,5,6], [7,8,9]])
arr2 = np.array([10,20,30])
print(arr1+arr2)
print(arr1-arr2)
print(arr1*arr2)
```

```
[[11 22 33]
 [14 25 36]
 [17 28 39]]
[[ -9 -18 -27]
 [ -6 -15 -24]
 [ -3 -12 -21]]
[[ 10  40  90]
 [ 40 100 180]
 [ 70 160 270]]
```

Aggregation Functions in NumPy

sum:

- Python numpy sum function calculates the sum of values in an array.

In [19]:
```
arr1 = np.array(10)
arr2 = np.array([20])
arr3 = np.array([23, 34, 56, 78])

print(arr1.sum())
print(arr2.sum())
print(arr3.sum())

print(np.sum(arr1))
print(np.sum(arr2))
print(np.sum(arr3))
```

```
10
20
191
10
20
191
```

- axis = 0 returns the sum of each column in a Numpy array.
- axis = 1 returns the sum of each row in a Numpy array.

In [20]:
```
arr = np.arange(1, 10, dtype='i').reshape(3, 3)

print(arr.sum(axis = 0))
print(arr.sum(axis = 1))

print(np.sum(arr, axis=0))
print(np.sum(arr, axis=1))
```

```
[12 15 18]
[ 6 15 24]
[12 15 18]
[ 6 15 24]
```

average:

- Python numpy average function returns the average of a given array.

In [21]:
```
arr = np.arange(1, 10, dtype='i').reshape(3, 3)

print(np.average(arr))
print(np.average(arr, axis=0))
print(np.average(arr, axis=1))
```

```
5.0
[4. 5. 6.]
[2. 5. 8.]
```

prod:

- Find product of all the elements in given array.

In [22]:
```
arr = np.arange(1, 10, dtype='i').reshape(3, 3)

print(np.prod(arr))
print(np.prod(arr, axis=0))
print(np.prod(arr, axis=1))
```

```
362880
[ 28  80 162]
[  6 120 504]
```

min:

In [23]:
```
arr = np.arange(11, 20, dtype='i').reshape(3, 3)

print(arr.min())
print(arr.min(axis=0))
print(arr.min(axis=1))

print(np.min(arr))
print(np.min(arr, axis=0))
print(np.min(arr, axis=1))
```

```
11
[11 12 13]
[11 14 17]
11
[11 12 13]
[11 14 17]
```

Array minimum:

- Python array minimum function accepts two arrays.
- Numpy array minimum performs one to one comparison of each array item in one array with other and returns an array of minimum values.

In [24]:
```
arr1 = np.array([12, 56, 34, 89, 10, 22, 94])
arr2 = np.array([45, 90, 23, 81, 98, 45, 34])

print(np.minimum(arr1, arr2))
```

```
[12 56 23 81 10 22 34]
```

Python array minimum function on randomly generated Matrices:

In [25]:
```
x = np.random.randint(1, 10, size = (5, 5))
print(x)
print()

y = np.random.randint(1, 10, size = (5, 5))
print(y)

print('\n-----Minimum Array----')
print(np.minimum(x, y))
```

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

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

-----Minimum Array----
[[4 1 5 2 5]
 [2 1 3 5 1]
 [3 1 8 5 4]
 [7 5 1 2 7]
 [5 1 2 4 6]]
```

max:

- Returns the maximum number from a given array or in a given axis.

In [26]:
```
arr = np.arange(11, 20, dtype='i').reshape(3, 3)

print(arr.max())
print(arr.max(axis=0))
print(arr.max(axis=1))

print(np.max(arr))
print(np.max(arr, axis=0))
print(np.max(arr, axis=1))
```

```
19
[17 18 19]
[13 16 19]
19
[17 18 19]
[13 16 19]
```

Array maximum:

In [27]:
```
arr1 = np.array([12, 56, 34, 89, 10, 22, 94])
arr2 = np.array([45, 90, 23, 81, 98, 45, 34])

print(np.maximum(arr1, arr2))
```

```
[45 90 34 89 98 45 94]
```

In [28]:
```
x = np.random.randint(1, 10, size = (5, 5))
print(x)
print()

y = np.random.randint(1, 10, size = (5, 5))
print(y)

print('\n-----Maximum Array----')
print(np.maximum(x, y))
```

```
[[7 9 6 7 7]
 [2 9 8 7 9]
 [6 2 6 2 3]
 [9 2 2 7 1]
 [1 6 8 4 3]]

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

-----Maximum Array----
[[7 9 8 7 9]
 [2 9 8 7 9]
 [6 2 6 7 3]
 [9 2 4 9 2]
 [2 6 8 5 7]]
```

mean:

- Returns the mean or average of a given array or in a given axis.

In [29]:
```
arr = np.arange(11, 20, dtype='i').reshape(3, 3)

print(arr.mean())
print(arr.mean(axis=0))
print(arr.mean(axis=1))

print(np.mean(arr))
print(np.mean(arr, axis=0))
print(np.mean(arr, axis=1))
```

```
15.0
[14. 15. 16.]
[12. 15. 18.]
15.0
[14. 15. 16.]
[12. 15. 18.]
```

median:

- Return the median of an array or an axis.

In [30]:
```
arr = np.arange(11, 20, dtype='i').reshape(3, 3)

print(np.median(arr))
print(np.median(arr, axis=0))
print(np.median(arr, axis=1))
```

```
15.0
[14. 15. 16.]
[12. 15. 18.]
```

numpy var function:

- The Python numpy var function returns the variance of a given array or in a given axis.
- The formula for this Python numpy var is :

```
(item1 – mean)2 + …(itemN – mean)2 / total items
```

In [31]:
```
arr = np.arange(11, 20, dtype='i').reshape(3, 3)

print(arr.var())
print(arr.var(axis=0))
print(arr.var(axis=1))

print(np.var(arr))
print(np.var(arr, axis=0))
print(np.var(arr, axis=1))
```

```
6.666666666666667
[6. 6. 6.]
[0.66666667 0.66666667 0.66666667]
6.666666666666667
[6. 6. 6.]
[0.66666667 0.66666667 0.66666667]
```

NumPy std:

- The Python numpy std function returns the standard deviation of a given array or in a given axis.
- The formula behind this is the numpy array square root of variance.

In [32]:
```
arr = np.arange(11, 20, dtype='i').reshape(3, 3)

print(arr.std())
print(arr.std(axis=0))
print(arr.std(axis=1))

print(np.std(arr))
print(np.std(arr, axis=0))
print(np.std(arr, axis=1))
```

```
2.581988897471611
[2.44948974 2.44948974 2.44948974]
[0.81649658 0.81649658 0.81649658]
2.581988897471611
[2.44948974 2.44948974 2.44948974]
[0.81649658 0.81649658 0.81649658]
```

numpy cumsum:

- Returns the cumulative sum of a given array or in a given axis.

In [33]:
```
arr = np.arange(11, 20, dtype='i').reshape(3, 3)

print(arr.cumsum())
print(arr.cumsum(axis=0))
print(arr.cumsum(axis=1))

print(np.cumsum(arr))
print(np.cumsum(arr, axis=0))
print(np.cumsum(arr, axis=1))
```

```
[ 11  23  36  50  65  81  98 116 135]
[[11 12 13]
 [25 27 29]
 [42 45 48]]
[[11 23 36]
 [14 29 45]
 [17 35 54]]
[ 11  23  36  50  65  81  98 116 135]
[[11 12 13]
 [25 27 29]
 [42 45 48]]
[[11 23 36]
 [14 29 45]
 [17 35 54]]
```

numpy cumprod:

- Returns the cumulative product of a given array or in a given axis.

In [34]:
```
arr = np.arange(11, 20, dtype='i').reshape(3, 3)

print(arr.cumprod())
print(arr.cumprod(axis=0))
print(arr.cumprod(axis=1))

print(np.cumprod(arr))
print(np.cumprod(arr, axis=0))
print(np.cumprod(arr, axis=1))
```

```
[        11        132       1716      24024     360360    5765760
   98017920 1764322560 -837609728]
[[  11   12   13]
 [ 154  180  208]
 [2618 3240 3952]]
[[  11  132 1716]
 [  14  210 3360]
 [  17  306 5814]]
[        11        132       1716      24024     360360    5765760
   98017920 1764322560 -837609728]
[[  11   12   13]
 [ 154  180  208]
 [2618 3240 3952]]
[[  11  132 1716]
 [  14  210 3360]
 [  17  306 5814]]
```

percentile:

- Finds the percentile (based on the given value) of an array or an axis.

In [35]:
```
arr1 = np.array([10, 20, 30, 40, 50])

print(np.percentile(arr1, 100))
print(np.percentile(arr1, 10))
```

```
50.0
14.0
```

argmin:

- Returns the index position of the minimum value in a given array or a given axis.

In [37]:
```
arr1 = np.array([12, 56, 34, 89, 10, 22, 94])
arr2 = np.array([45, 90, 23, 81, 98, 45, 34])

print(np.argmin(arr1))
print(np.argmin(arr2))

print(np.argmin(arr1, axis=0))
print(np.argmin(arr2, axis=0))
```

```
4
2
4
2
```

argmax:

- Returns the index position of the maximum value in a given array or a given axis.

In [38]:
```
arr1 = np.array([12, 56, 34, 89, 10, 22]).reshape(3,2)
arr2 = np.array([45, 90, 23, 81, 98, 45]).reshape(3,2)

print(np.argmax(arr1))
print(np.argmax(arr2))

print(np.argmax(arr1, axis=0))
print(np.argmax(arr2, axis=0))

print(np.argmax(arr1, axis=1))
print(np.argmax(arr2, axis=1))
```

```
3
4
[1 1]
[2 0]
[1 1 1]
[1 1 0]
```

corrcoef:

- Numpy corrcoef function find and returns the correlation coefficient of an array.

In [39]:
```
arr1 = np.array([12, 45, 78, 90, 83, 94, 41, 28, 49])
arr2 = np.array([56, 43, 98, 32, 41, 88, 99, 30, 20])

print(np.corrcoef(arr1))
print(np.corrcoef(arr2))
```

```
1.0
1.0
```

Linear Algebra(LinAlg) Functions

- Linear Algebra module of NumPy offers various methods to apply linear algebra on any numpy array.

matrix_rank:

In [40]:
```
a = np.array([[6, 1, 1], [4, -2, 5], [2, 8, 7]])

print("Rank of a:", np.linalg.matrix_rank(a))
```

```
Rank of a: 3
```

trace:

In [41]:
```
print("\nTrace of A:", np.trace(a))
```

```
Trace of A: 11
```

determinent:

In [42]:
```
print("\nDeterminant of A:", np.linalg.det(a))
```

```
Determinant of A: -306.0
```

inverse:

In [43]:
```
print("\nInverse of A:\n", np.linalg.inv(a))
```

```
Inverse of A:
 [[ 0.17647059 -0.00326797 -0.02287582]
 [ 0.05882353 -0.13071895  0.08496732]
 [-0.11764706  0.1503268   0.05228758]]
```

power of Matrix:

In [44]:
```
print("\nMatrix A raised to power 3:\n", np.linalg.matrix_power(a, 3))
```

```
Matrix A raised to power 3:
 [[336 162 228]
 [406 162 469]
 [698 702 905]]
```

Eigen Values:

- Return the eigenvalues and eigenvectors of a complex Hermitian (conjugate symmetric) or a real symmetric matrix.

- Returns two objects, a 1-D array containing the eigenvalues of a, and a 2-D square array or matrix (depending on the input type) of the corresponding eigenvectors (in

In [45]:
```
c, d = np.linalg.eigh(a)

print(c, d)
```

```
[-7.15957544  4.56008699 13.59948845] [[ 0.192948   -0.86902847 -0.455588
18]
 [-0.86570993  0.06778288 -0.49593528]
 [ 0.46186296  0.49009693 -0.739248  ]]
```

Matrix Product:

- Returns the dot product of vectors a and b.
- It can handle 2D arrays but considering them as matrix and will perform matrix multiplication.

In [47]:
```
vector_a = 2 + 3j
vector_b = 4 + 5j

product = np.dot(vector_a, vector_b)
print("Dot Product  : ", product)
```

```
Dot Product  :  (-7+22j)
```

In [ ]:

In [ ]: