In [81]:
```
import numpy as np
```

In [42]:
```
x = np.array([[-1,2,-3,4,5]])
y = np.array([[1.1,1.2,1.3,1.55,9.66667]])
z = np.array([1,2,3,4,5])
a = np.array([0,0,0,0])
```

In [109…
```
##task 4
#abs , fabs , sqrt , square , exp , log , sign , ceil , floor , rint , modf, isnan , i
#cos , sin , tan and their hyperbolic , arc[sin , cos , tan , sinh , cosh , tanh] , lo
#abs return a integer array
#fabs returns a float array
#sqrt function find the square root of an array if array contains negative number then
#square funtion finds the square of each element in an array
#exp calculates the exponent of each element in an array
#log find the log of each element
#sign find the sign of each element 1 represents postive , 0 zero , -1 negative
#ceil rounds off to next value even int has .1 decimal part
#floor remove the floating point part and return int
#rint rounds off mathematically
#modf return deciaml and integer part separtly
#isnan returns ture if value is not a number
#isfinite return ture if value is finite
#isinf return ture if value is infinite
#sin , cos , tan and their hyperbolic reutrn their trigonomerical functional value
#arc[sin , cos ,tan , hyperbolic] return their inverse trignometrical
result = np.abs(x)

print(np.fabs(result))

print(np.sqrt(result))

print(np.square(result))

print(np.exp(result))

print(np.log(result))

print(np.sign(result))

print(np.ceil(y))

print(np.floor(y))

print(np.rint(y))

print(np.modf(y))

print(np.isnan(z))

print(np.isfinite(z))

print(np.isinf(z))

print(np.sin(z))
print(np.cos(z))
print(np.tan(z))
print(np.sinh(z))
print(np.cosh(z))
print(np.tanh(z))

print(np.arccos(a))
print(np.arcsin(a))
print(np.arctan(a))
print(np.arcsinh(a))

print(np.logical_not(a))
```

```
[[1. 2. 3. 4. 5.]]
[[1.         1.41421356 1.73205081 2.         2.23606798]]
[[ 1  4  9 16 25]]
[[  2.71828183   7.3890561   20.08553692  54.59815003 148.4131591 ]]
[[0.         0.69314718 1.09861229 1.38629436 1.60943791]]
[[1 1 1 1 1]]
[[ 2.  2.  2.  2. 10.]]
[[1. 1. 1. 1. 9.]]
[[ 1.  1.  1.  2. 10.]]
(array([[0.1    , 0.2    , 0.3    , 0.55   , 0.66667]]), array([[1., 1., 1., 1.,
9.]]))
[False False False False False]
[ True  True  True  True  True]
[False False False False False]
[ 0.84147098  0.90929743  0.14112001 -0.7568025  -0.95892427]
[ 0.54030231 -0.41614684 -0.9899925  -0.65364362  0.28366219]
[ 1.55740772 -2.18503986 -0.14254654  1.15782128 -3.38051501]
[ 1.17520119  3.62686041 10.01787493 27.2899172  74.20321058]
[ 1.54308063  3.76219569 10.067662   27.30823284 74.20994852]
[0.76159416 0.96402758 0.99505475 0.9993293  0.9999092 ]
[1.57079633 1.57079633 1.57079633 1.57079633]
[0. 0. 0. 0.]
[0. 0. 0. 0.]
[0. 0. 0. 0.]
[ True  True  True  True]
```

In [61]:

```
#task 5
#add , subtact , multiply , divide , floor divide , power , maximum , fmax , minimum ,
print(np.add(x,1))
print(np.subtract(x,1))
print(np.multiply(x,2))
print(np.divide(x,1))
print(np.floor_divide(x,2))
print(np.power(x,2))

print(np.maximum(x,3))
print(np.fmax(x,1))
print(np.minimum(x,y))
print(np.fmin(x,2))
print(np.mod(x,2))
```

```
[[ 0  3 -2  5  6]]
[[-2  1 -4  3  4]]
[[-2  4 -6  8 10]]
[[-1.  2. -3.  4.  5.]]
[[-1  1 -2  2  2]]
[[ 1  4  9 16 25]]
ma [[3 3 3 4 5]]
[[1 2 1 4 5]]
[[-1.    1.2  -3.    1.55  5.  ]]
[[-1  2 -3  2  2]]
[[1 0 1 0 1]]
```

In [65]:
```
#task 6
print(np.sum(x))
print(np.mean(x))
print(np.std(x))
print(np.var(x))
print(np.min(x))
print(np.max(x))
print(np.argmin(x))
print(np.argmax(x))
print(np.cumsum(z))
print(np.cumprod(z))
```

```
7
1.4
3.006659275674582
9.040000000000001
-3
5
2
4
[ 1  3  6 10 15]
[  1   2   6  24 120]
```

In [71]:
```
#task 7
print(np.unique(z))
print(np.intersect1d(z , y))
print(np.union1d(z , y))
print(np.in1d(z , y))
print(np.setdiff1d(z , y))
print(np.setxor1d(z , y))
```

```
[1 2 3 4 5]
[]
[1.      1.1     1.2     1.3     1.55    2.      3.      4.      5.
 9.66667]
[False False False False False]
[1 2 3 4 5]
[1.      1.1     1.2     1.3     1.55    2.      3.      4.      5.
 9.66667]
```

In [106…
```
#task 8
squaremat = np.arange(1,26).reshape(5,5)
squaremat2 = np.arange(1,26).reshape(5,5)

print(np.diag(squaremat))
print(np.dot(squaremat , squaremat2))
print(np.trace(squaremat))
print(np.linalg.eig(squaremat))
```

```
[ 1  7 13 19 25]
[[ 215  230  245  260  275]
 [ 490  530  570  610  650]
 [ 765  830  895  960 1025]
 [1040 1130 1220 1310 1400]
 [1315 1430 1545 1660 1775]]
65
(array([ 6.86420807e+01+0.00000000e+00j, -3.64208074e+00+0.00000000e+00j,
       -1.04866446e-15+0.00000000e+00j,  1.34082976e-16+1.19171295e-15j,
        1.34082976e-16-1.19171295e-15j]), array([[-0.10797496+0.j        ,  0.6749528
3+0.j        ,
         0.02031966+0.j        , -0.24674761-0.00953463j,
        -0.24674761+0.00953463j],
       [-0.25277499+0.j        ,  0.3603897 +0.j        ,
         0.1802646 +0.j        ,  0.08248136+0.28769623j,
         0.08248136-0.28769623j],
       [-0.39757502+0.j        ,  0.04582657+0.j        ,
         0.10205537+0.j        ,  0.05755382-0.41247509j,
         0.05755382+0.41247509j],
       [-0.54237506+0.j        , -0.26873656+0.j        ,
        -0.82618318+0.j        ,  0.62443868+0.j        ,
         0.62443868-0.j        ],
       [-0.68717509+0.j        , -0.58329969+0.j        ,
         0.52354355+0.j        , -0.51772627+0.13431349j,
        -0.51772627-0.13431349j]]))
```

In [108…

```
#task 9
np.random.seed(7)
print(np.random.randint(10 , size = (5,3)))
print(np.random.permutation(x))
print(np.random.shuffle(x))
```

```
[[4 9 6]
 [3 3 7]
 [7 9 7]
 [8 9 8]
 [7 6 4]]
[[-1  2 -3  4  5]]
None
```