In [3]:
```
import numpy as np
```

In [4]:
```
#this is a function to compute reciprocal of each element of an array
def reciprocal(arr):
    output=np.empty(len(arr))
    for i in range(len(arr)):
        output[i]=1/a[i]
    return output
```

In [5]:
```
#timeit function is used to calculate the time take per loop
a=np.arange(1,1000000)#array with approximately 1 million elements
%timeit reciprocal(a)
```

```
631 ms ± 20.4 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

In [7]:
```
%timeit 1/a#time take is much less when we use vectorised function
```

```
6.28 ms ± 207 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

In [15]:
```
ar=np.arange(5)
print(ar)
ar1=np.arange(1,6)
print(ar1)
print(ar/ar1)#division of two arrays means corresponding elements are divided
```

```
[0 1 2 3 4]
[1 2 3 4 5]
[0.         0.5        0.66666667 0.75       0.8       ]
```

In [18]:
```
x = np.arange(9).reshape((3, 3))
print(x)
print(2**x)#each element is the power of two
```

```
[[0 1 2]
 [3 4 5]
 [6 7 8]]
[[  1   2   4]
 [  8  16  32]
 [ 64 128 256]]
```

In [42]:
```
a=np.arange(4)
print(a)#original array
print(a+2)#each element 2 is added
print(a-1)#each element is subtracted by 1
print(a*3)#each element is multiplied by 3
print(a/3)#each element is divided by 3
a1=np.arange(10)
print(a1)
print(a1//4)#each element is divided by 4 floor division
print(-a1)#negative of given array
print(2**a1)#each element 2 power that element
print(a1**2)#squaring each element
print(a1%2)#modular division of 2 of each element
#numpy arithmetic functions
a1=np.arange(5)
print(a1)
np.add(a1,2)#add 2 from each element
np.subtract(a1,2)#subtract 2 from each element
np.negative(a1)#neagtive of array
np.multiply(a1,3)#multiply each element from 3
np.divide(a1,2)#divide each element by 5
np.floor_divide(a1,2)
np.mod(a1,2)
```

```
[0 1 2 3]
[2 3 4 5]
[-1  0  1  2]
[0 3 6 9]
[0.         0.33333333 0.66666667 1.        ]
[0 1 2 3 4 5 6 7 8 9]
[0 0 0 0 1 1 1 1 2 2]
[ 0 -1 -2 -3 -4 -5 -6 -7 -8 -9]
[  1   2   4   8  16  32  64 128 256 512]
[ 0  1  4  9 16 25 36 49 64 81]
[0 1 0 1 0 1 0 1 0 1]
[0 1 2 3 4]
```

Out[42]:
```
array([0, 1, 0, 1, 0], dtype=int32)
```

In [43]:
```
#absloute function
a=np.array([1,-2,-3,4,-5,6,7,-8,9,0])
print(abs(a))#converts each negative number to positive numbers
```

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

In [52]:
```
np.absolute(a)
#for complex variables absolute returns magnitude
c=np.array([10+1j,12,3+4j,7j])
np.absolute(c)
```

Out[52]:
```
array([10.04987562, 12.        ,  5.        ,  7.        ])
```

In [56]:
```
#trigonometric functions
theta=np.linspace(0,np.pi,3)
print(theta)#printing theta list
print(np.sin(theta))#printing sine theta
print(np.cos(theta))#printing cos theta
print(np.tan(theta))#printing tan theta
```

```
[0.         1.57079633 3.14159265]
[0.0000000e+00 1.0000000e+00 1.2246468e-16]
[ 1.000000e+00  6.123234e-17 -1.000000e+00]
[ 0.00000000e+00  1.63312394e+16 -1.22464680e-16]
```

In [60]:
```
thetha=np.array([-1,0,1])
print(np.arcsin(thetha))#sin inverse
print(np.arccos(thetha))#cos inverse
print(np.arctan(thetha))#tan inverse
```

```
[-1.57079633  0.          1.57079633]
[3.14159265 1.57079633 0.        ]
[-0.78539816  0.          0.78539816]
```

In [17]:
```
#exponents and logorithmic functions
import numpy as np
p=np.array([1,2,3,4])
print("e power x:",np.exp(p))
print("e power x -1",np.expm1(p))#expontential minus 1
print("2 power x:",np.exp2(p))#constant to the power of each element in array
print(np.power(p,2))#squaring each element using power function
x=np.array([2,4,8,16])
print("x=",x)
print("lnx=",np.log(x))#defualt logarithm with base e
print("log to the base 2:",np.log2(x))#logarithm with base 2
print("log(1+x)=",np.log1p(x))#logarithm of 1 plus each element of array
#log1p and expm1 are used for very precise values
```

```
e power x: [ 2.71828183  7.3890561  20.08553692 54.59815003]
e power x -1 [ 1.71828183  6.3890561  19.08553692 53.59815003]
2 power x: [ 2.  4.  8. 16.]
[ 1  4  9 16]
x= [ 2  4  8 16]
lnx= [0.69314718 1.38629436 2.07944154 2.77258872]
log to the base 2: [1. 2. 3. 4.]
log(1+x)= [1.09861229 1.60943791 2.19722458 2.83321334]
```

In [26]:
```
#scipy module is used to import specialised ufuncs
from scipy import special
x=[1,2,3]
print("x:",x)
print("gamma(x):",special.gamma(x))#gamma fuction
print("ln(gamma(x)):",special.gammaln(x))#ln of gamma function
#guassian integrals or error function
print("gaussian function:",special.erf(x))
print("complement of guassian function:",special.erfc(x))#compliment of error function
print("inverse of guassian function:",special.erfinv(x))#inverse of error function
```

```
x: [1, 2, 3]
gamma(x): [1. 1. 2.]
ln(gamma(x)): [0.         0.         0.69314718]
gaussian function: [0.84270079 0.99532227 0.99997791]
complement of guassian function: [1.57299207e-01 4.67773498e-03 2.20904970e-05]
inverse of guassian function: [inf nan nan]
```

In [32]:
```
#advanced numpy functions
#out is used for directly assigning where the computated result is stored instead of creating a new array
x=np.arange(0,10)
print("x:",x)
y=np.empty(10)
np.power(2,x,out=y)
print("2 to the power of each element in x:",y)
```

```
x: [0 1 2 3 4 5 6 7 8 9]
2 to the power of each element in x: [  1.   2.   4.   8.  16.  32.  64. 128. 256. 512.]
```

In [42]:
```
#aggreagates in numpy
a=np.arange(1,11)
print("a=",a)
#reduce will kepp on performing operations
print("addition of all elements:",np.add.reduce(a))#addition of all elements
print("mulitplication of all elements:",np.multiply.reduce(a))#multiplication of all elements
```

```
a= [ 1  2  3  4  5  6  7  8  9 10]
addition of all elements: 55
mulitplication of all elements: 3628800
```

In [43]:
```
#accumulate is used for getting intermediates
np.add.accumulate(a)#intermediate sums will be displayed
np.multiply.accumulate(a)#intermediate product will be displayed
```

Out[43]:
```
array([ 1,  3,  6, 10, 15, 21, 28, 36, 45, 55])
```

In [48]:
```
#outer method
#computes all pairs of two different inputs
a=np.arange(1,11)
print("a=",a)
print("mulitplication of all poissible combination of pairs:")
print(np.multiply.outer(a,a))
```

```
a= [ 1  2  3  4  5  6  7  8  9 10]
mulitplication of all poissible combination of pairs:
[[  1   2   3   4   5   6   7   8   9  10]
 [  2   4   6   8  10  12  14  16  18  20]
 [  3   6   9  12  15  18  21  24  27  30]
 [  4   8  12  16  20  24  28  32  36  40]
 [  5  10  15  20  25  30  35  40  45  50]
 [  6  12  18  24  30  36  42  48  54  60]
 [  7  14  21  28  35  42  49  56  63  70]
 [  8  16  24  32  40  48  56  64  72  80]
 [  9  18  27  36  45  54  63  72  81  90]
 [ 10  20  30  40  50  60  70  80  90 100]]
```