```
import numpy as np
```

```
arr = np.arange(0,11)
```

```
arr
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

```
arr[8]
```

```
8
```

```
arr[1:5]
```

```
array([1, 2, 3, 4])
```

```
arr[0:5]
```

```
array([0, 1, 2, 3, 4])
```

```
arr[0:5]=100
arr
```

```
array([100, 100, 100, 100, 100,   5,   6,   7,   8,   9,  10])
```

```
arr = np.arange(0,11)
arr
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

```
slice_of_arr = arr[0:6]
slice_of_arr
```

```
array([0, 1, 2, 3, 4, 5])
```

```
slice_of_arr[:]=99
slice_of_arr
```

```
array([99, 99, 99, 99, 99, 99])
```

```
arr
```

```
array([99, 99, 99, 99, 99, 99,  6,  7,  8,  9, 10])
```

```
arr_copy = arr.copy()
arr_copy
```

```
array([99, 99, 99, 99, 99, 99,  6,  7,  8,  9, 10])
```

```
arr_2d = np.array(([5,10,15],[20,25,30],[35,40,45]))
arr_2d
```

```
array([[ 5, 10, 15],
       [20, 25, 30],
       [35, 40, 45]])
```

```
arr_2d[1]
```

```
array([20, 25, 30])
```

```
arr_2d[1,0]
```

```
20
```

```
arr_2d[:2,1:]
```

```
array([[10, 15],
       [25, 30]])
```

```
arr_2d[2]
```

```
array([35, 40, 45])
```

```
arr_2d[2,:]
```

```
array([35, 40, 45])
```

```
arr2d=np.zeros((10,10))
```

```
arr_length=arr2d.shape[1]
```

```
for i in range(arr_length):
  arr2d[i]=i
arr2d
```

```
array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],
       [2., 2., 2., 2., 2., 2., 2., 2., 2., 2.],
       [3., 3., 3., 3., 3., 3., 3., 3., 3., 3.],
       [4., 4., 4., 4., 4., 4., 4., 4., 4., 4.],
       [5., 5., 5., 5., 5., 5., 5., 5., 5., 5.],
       [6., 6., 6., 6., 6., 6., 6., 6., 6., 6.],
       [7., 7., 7., 7., 7., 7., 7., 7., 7., 7.],
       [8., 8., 8., 8., 8., 8., 8., 8., 8., 8.],
       [9., 9., 9., 9., 9., 9., 9., 9., 9., 9.]])
```

```
arr2d[[2,4,6,8]]
```

```
array([[2., 2., 2., 2., 2., 2., 2., 2., 2., 2.],
       [4., 4., 4., 4., 4., 4., 4., 4., 4., 4.],
       [6., 6., 6., 6., 6., 6., 6., 6., 6., 6.],
       [8., 8., 8., 8., 8., 8., 8., 8., 8., 8.]])
```

```
arr2d[[6,4,2,7]]
```

```
array([[6., 6., 6., 6., 6., 6., 6., 6., 6., 6.],
       [4., 4., 4., 4., 4., 4., 4., 4., 4., 4.],
       [2., 2., 2., 2., 2., 2., 2., 2., 2., 2.],
       [7., 7., 7., 7., 7., 7., 7., 7., 7., 7.]])
```

```
arr =np.arange(1,11)
arr
```

```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

```
arr > 4
```

```
array([False, False, False, False,  True,  True,  True,  True,  True,
        True])
```

```
bool_arr = arr>4
```

```
bool_arr
```

```
array([False, False, False, False,  True,  True,  True,  True,  True,
        True])
```

```
arr[bool_arr]
```

```
array([ 5,  6,  7,  8,  9, 10])
```

```
arr[arr>2]
```

```
array([ 3,  4,  5,  6,  7,  8,  9, 10])
```

```
x=2
arr[arr>x]
```

```
array([ 3,  4,  5,  6,  7,  8,  9, 10])
```

```
my_list= [1,2,3,4,]
print(my_list)
print(type(my_list))
```

```
[1, 2, 3, 4]
<class 'list'>
```

```
import numpy as np
```

```
arr=np.array(my_list)
print(arr)
print(type(arr))
```

```
[1 2 3 4]
<class 'numpy.ndarray'>
```

```
my_matrix=[[1,2,3],[4,5,6],[7,8,9]]
print(my_matrix)
print(type(my_matrix))
```

```
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
<class 'list'>
```

```
np.array(my_matrix)
```

```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

```
print(np.array(my_matrix))
```

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

```
print(type(np.array(my_matrix)))
```

```
<class 'numpy.ndarray'>
```

```
np.arange(0,10)
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
np.arange(2,20)
```

```
array([ 2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17, 18,
       19])
```

```
np.arange(my_matrix(1,5))
```

```
np.arange(0,10,3)
```

```
array([0, 3, 6, 9])
```

```
np.zeros(5)
```

```
array([0., 0., 0., 0., 0.])
```

```
np.zeros((4,4))
```

```
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
```

```
np.zeros((6,4,2))
```

```
array([[[0., 0.],
        [0., 0.],
        [0., 0.],
        [0., 0.]],

       [[0., 0.],
        [0., 0.],
        [0., 0.],
        [0., 0.]],

       [[0., 0.],
        [0., 0.],
        [0., 0.],
        [0., 0.]],

       [[0., 0.],
        [0., 0.],
        [0., 0.],
        [0., 0.]],

       [[0., 0.],
        [0., 0.],
        [0., 0.],
        [0., 0.]],

       [[0., 0.],
        [0., 0.],
        [0., 0.],
        [0., 0.]]])
```

```
np.ones((1,4,4))
```

```
array([[[1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.]]])
```

Double-click (or enter) to edit

```
np.linspace(0,10,4)
```

```
array([ 0.        ,  3.33333333,  6.66666667, 10.        ])
```

```
np.linspace(0,10,5)
```

```
array([ 0. ,  2.5,  5. ,  7.5, 10. ])
```

```
np.eye(50)
```

```
array([[1., 0., 0., ..., 0., 0., 0.],
       [0., 1., 0., ..., 0., 0., 0.],
       [0., 0., 1., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 1., 0., 0.],
       [0., 0., 0., ..., 0., 1., 0.],
       [0., 0., 0., ..., 0., 0., 1.]])
```

```
np.random.rand(6)
```

```
array([0.98915084, 0.72568535, 0.11748075, 0.30114676, 0.17622066,
       0.18720565])
```

```
np.random.rand(5,5,2)
```

```
array([[[0.60351398, 0.05738968],
        [0.68463086, 0.91047196],
        [0.95528408, 0.21309584],
        [0.06999391, 0.1096453 ],
        [0.37403682, 0.55844375]],

       [[0.6304985 , 0.21246983],
        [0.89700289, 0.02771646],
        [0.97701754, 0.49827104],
        [0.72549847, 0.78357863],
        [0.55785172, 0.40480014]],

       [[0.76917717, 0.19518931],
```

```
        [0.13264183, 0.1258713 ],
        [0.0898754 , 0.19404914],
        [0.68006675, 0.85185722],
        [0.57372115, 0.67512099]],

       [[0.81543839, 0.19692475],
        [0.4787941 , 0.25606791],
        [0.21996484, 0.60209123],
        [0.21814538, 0.36855781],
        [0.56993201, 0.59814626]],

       [[0.92783609, 0.5887963 ],
        [0.35097801, 0.92434874],
        [0.91624876, 0.82582402],
        [0.7962011 , 0.38193032],
        [0.1609569 , 0.91540096]]]])
```

```
np.random.randn(2,5,5)
```

```
array([[[-0.09874752,  0.0081019 , -1.06659521,  0.13693104,
          0.48704131],
        [-1.20296068,  0.76106879, -1.10141479,  0.90131306,
          0.39517262],
        [-0.35286571,  0.92867196,  1.20487235,  0.4863117 ,
         -1.2545478 ],
        [-0.96002663, -0.8317218 , -0.25667751,  2.62579549,
          0.36674329],
        [-0.55286845, -0.28404729,  0.38166566,  1.96518571,
         -1.42443341]],

       [[-1.37730492,  1.05476512, -0.25371845,  2.54017378,
          0.01136691],
        [-1.2862719 ,  0.93558455, -1.26255692,  0.9949774 ,
         -1.21006503],
        [ 1.29106346,  0.21554628, -0.66498616,  0.5402126 ,
         -0.36165217],
        [ 0.69728301,  2.09036269, -0.15663154,  1.47595769,
         -1.6410172 ],
        [-1.10312392, -0.91606701, -1.60780256, -0.52402702,
          0.40005345]]])
```

```
np.random.randint(1,4,13)
```

```
array([2, 1, 3, 3, 1, 1, 3, 2, 3, 2, 2, 1, 2])
```

```
arr=np.arange(5,10,5)
print(arr)
```

```
[5]
```

```
arr=np.random.randint(0,50,24)
print(arr)
```

```
[20  8 13 49  9  1 29 22 38 38 29 37 11  2 33  4  0 24 17  0 24  0 47 38]
```

```
arr
```

```
array([20,  8, 13, 49,  9,  1, 29, 22, 38, 38, 29, 37, 11,  2, 33,  4,  0,
       24, 17,  0, 24,  0, 47, 38])
```

```
ar
```

```
array([25, 16,  7, 16, 41,  2, 41,  9, 37,  7,  8, 47, 48, 43, 23, 31, 40,
       20, 24,  9, 48, 11,  3])
```

```
arr.reshape(4,6)
```

```
array([[26, 32, 16, 47, 23, 47],
       [36,  0, 21, 40,  6, 13],
       [10, 21, 11, 13, 38, 21],
       [33, 23, 22,  7, 30, 41]])
```

```
arr
```

```
array([20,  8, 13, 49,  9,  1, 29, 22, 38, 38, 29, 37, 11,  2, 33,  4,  0,
       24, 17,  0, 24,  0, 47, 38])
```

```
arr.max()
```

```
47
```

```
arr.min()
```

```
0
```

```
arr.argmin()
```

```
7
```

```
arr.shape
```

```
(24,)
```

```
arr.reshape(4,6)
```

```
array([[26, 32, 16, 47, 23, 47],
       [36,  0, 21, 40,  6, 13],
       [10, 21, 11, 13, 38, 21],
       [33, 23, 22,  7, 30, 41]])
```

```
arr.reshape(4,6)
```

```
array([[20,  8, 13, 49,  9,  1],
       [29, 22, 38, 38, 29, 37],
       [11,  2, 33,  4,  0, 24],
       [17,  0, 24,  0, 47, 38]])
```

```
arr.dtype
```

```
dtype('int64')
```

```
arr.shape
```

```
(24,)
```

```
import numpy as np
```

```
a= np.array([1,2,3,4,5,6,7,8])
a[0:2]
```

```
array([1, 2])
```

```
a[3:7]
```

```
array([4, 5, 6, 7])
```

```
a[0:-6]
```

```
array([1, 2])
```

```
a[0:]
```

```
array([1, 2, 3, 4, 5, 6, 7, 8])
```

```
a[-4:-2]
```

```
array([5, 6])
```

```
a[:]
```

```
array([1, 2, 3, 4, 5, 6, 7, 8])
```

```
a[::2]
```

```
array([1, 3, 5, 7])
```

```
a[::-3]
```

```
array([8, 5, 2])
```

```
x="Welcome to SASI Engineering Collage"
print(x[0:7])
print(len(x))
```

```
Welcome
35
```

```
print(x[11:])
```

```
SASI Engineering Collage
```

```
print(x[::-1])
```

```
egalloC gnireenignE ISAS ot emocleW
```

```
x_x=slice(1,30,4)
print(a[x_x])
```

```
[2]
```

```
import numpy as np
```

```
a=np.array([1,2,3,4,5,6,7,8])
a[7:]
```

```
array([8])
```

```
a[0:-4]
```

```
array([1, 2, 3, 4])
```

```
a[:]
```

```
array([1, 2, 3, 4, 5, 6, 7, 8])
```

```
a[::7]
```

```
array([1, 8])
```

```
a[::-2]
```

```
array([8, 6, 4, 2])
```

```
a[-2::-2]
```

```
array([7, 5, 3, 1])
```

```
str="Welcome to SASI Engineering collage"
print(str[0])
print(len(str))
```

```
W
35
```

```
print(str[11::])
```

```
SASI Engineering collage
```

```
print(str[::-1])
```

```
egalloc gnireenignE ISAS ot emocleW
```

```
mysc=slice(1,5,2)
print(a[mysc])
```

```
[2 4]
```

```
mysc=slice(-1,-12,-1)
print(str[mysc])
```

```
egalloc gni
```

```
s=input("Enter a string :")
sub=input("Enter a sub string :")
print(s.find(sub))
```

```
Enter a string :hello world
Enter a sub string :world
6
```

```
len(s)
```

```
11
```

Double-click (or enter) to edit

```
import statistics as s
print(s.mean([2,4,6]))
```

```
4
```

```
print(s.harmonic_mean([10,30,50,70,90]))
```

```
27.97513321492007
```

```
x=min(-5,-10,-2)
print(x)
```

```
-10
```

```
x=abs(-37)
print(x)
```

```
37
```

```
x=pow(-2,4)
print(x)
```

```
16
```

```
import math as m
x=m.sqrt(64)
print(x)
```

```
8.0
```

```
x=m.floor(1.4)
print(x)
```

```
1
```

```
x=m.ceil(1.4)
print(x)
```

```
2
```

```
x=m.pi
print(x)
```

```
3.141592653589793
```

```
import numpy as np
arr=np.arange(0,10)
arr
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
arr*arr
```

```
array([ 0,  1,  4,  9, 16, 25, 36, 49, 64, 81])
```

```
arr-arr
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

```
arr/arr
```

```
<ipython-input-18-50b4ced5627e>:1: RuntimeWarning: invalid value encountered in divide
  arr/arr
array([nan,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.])
```

```
arr**3
```

```
array([  0,   1,   8,  27,  64, 125, 216, 343, 512, 729])
```

```
np.sqrt(arr)
```

```
array([0.        , 1.        , 1.41421356, 1.73205081, 2.        ,
       2.23606798, 2.44948974, 2.64575131, 2.82842712, 3.        ])
```

```
np.exp(arr)
```

```
array([1.00000000e+00, 2.71828183e+00, 7.38905610e+00, 2.00855369e+01,
       5.45981500e+01, 1.48413159e+02, 4.03428793e+02, 1.09663316e+03,
       2.98095799e+03, 8.10308393e+03])
```

```
np.max(arr)
```

```
9
```

```
np.min(arr)
```

```
0
```

```
np.log(arr)
```

```
<ipython-input-24-a67b4ae04e95>:1: RuntimeWarning: divide by zero encountered in log
  np.log(arr)
array([      -inf, 0.        , 0.69314718, 1.09861229, 1.38629436,
       1.60943791, 1.79175947, 1.94591015, 2.07944154, 2.19722458])
```

```
arr1=np.array([[1,2,3],[4,5,6],[7,8,9]])
arr1
```

```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

```
arr2=[[1,2,3],[4,5,6],[7,8,9]]
arr2[0][2]
```

```
3
```

```
np.ones(5)
```

```
array([1., 1., 1., 1., 1.])
```

```
np.ones((2,2))
```

```
array([[1., 1.],
       [1., 1.]])
```

```
np.linspace(0,10,4)
```

```
array([ 0.        ,  3.33333333,  6.66666667, 10.        ])
```

```
np.linspace(0,10,50)
```

```
array([ 0.        ,  0.20408163,  0.40816327,  0.6122449 ,  0.81632653,
        1.02040816,  1.2244898 ,  1.42857143,  1.63265306,  1.83673469,
        2.04081633,  2.24489796,  2.44897959,  2.65306122,  2.85714286,
        3.06122449,  3.26530612,  3.46938776,  3.67346939,  3.87755102,
        4.08163265,  4.28571429,  4.48979592,  4.69387755,  4.89795918,
        5.10204082,  5.30612245,  5.51020408,  5.71428571,  5.91836735,
        6.12244898,  6.32653061,  6.53061224,  6.73469388,  6.93877551,
        7.14285714,  7.34693878,  7.55102041,  7.75510204,  7.95918367,
        8.16326531,  8.36734694,  8.57142857,  8.7755102 ,  8.97959184,
        9.18367347,  9.3877551 ,  9.59183673,  9.79591837, 10.        ])
```

```
np.eye(3)
```

```
array([[1., 0., 0.],
       [0., 1., 0.],
```

```
       [0., 0., 1.]])
```

```
np.random.rand(5)
```

```
array([0.75491569, 0.02355573, 0.70075466, 0.56924687, 0.41401132])
```

```
np.random.rand(6)
```

```
array([0.64398772, 0.11290953, 0.15670773, 0.97718162, 0.09777336,
       0.43828925])
```

```
np.random.rand(5,5)
```

```
array([[0.48737273, 0.30603841, 0.91344483, 0.60023343, 0.52585742],
       [0.06422271, 0.80393497, 0.03861707, 0.85451362, 0.1811598 ],
       [0.86297057, 0.05895708, 0.9468471 , 0.54808268, 0.78580546],
       [0.93176929, 0.39172821, 0.87545829, 0.29070762, 0.98246993],
       [0.06960282, 0.05994629, 0.6949342 , 0.85199657, 0.80296504]])
```

```
np.random.randn(2)
```

```
array([-1.66615377,  0.25808248])
```

```
np.random.randint(1,100)
```

```
38
```

```
np.random.randint(1,100,4)
```

```
array([94, 55, 34, 65])
```

```
arr3=np.arange(25)
arr3
```

```
array([ 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])
```

arr3.reshape(5,5)

```
import scipy
```

```
print(scipy.__version__)
```

```
1.11.3
```

```
from scipy import constants
help(constants)
```

```
Help on package scipy.constants in scipy:

NAME
    scipy.constants

DESCRIPTION
    ==================================
    Constants (:mod:`scipy.constants`)
    ==================================

    .. currentmodule:: scipy.constants

    Physical and mathematical constants and units.


    Mathematical constants
    ======================

    ================  =================================================================
    ``pi``            Pi
    ``golden``        Golden ratio
    ``golden_ratio``  Golden ratio
    ================  =================================================================
```

```
    Physical constants
    ==================

    ===========================  ================================================================
    ``c``                        speed of light in vacuum
    ``speed_of_light``           speed of light in vacuum
    ``mu_0``                     the magnetic constant :math:`\mu_0`
    ``epsilon_0``                the electric constant (vacuum permittivity), :math:`\epsilon_0`
    ``h``                        the Planck constant :math:`h`
    ``Planck``                   the Planck constant :math:`h`
    ``hbar``                     :math:`\hbar = h/(2\pi)`
    ``G``                        Newtonian constant of gravitation
    ``gravitational_constant``   Newtonian constant of gravitation
    ``g``                        standard acceleration of gravity
    ``e``                        elementary charge
    ``elementary_charge``        elementary charge
    ``R``                        molar gas constant
    ``gas_constant``             molar gas constant
    ``alpha``                    fine-structure constant
    ``fine_structure``           fine-structure constant
    ``N_A``                      Avogadro constant
    ``Avogadro``                 Avogadro constant
    ``k``                        Boltzmann constant
    ``Boltzmann``                Boltzmann constant
    ``sigma``                    Stefan-Boltzmann constant :math:`\sigma`
    ``Stefan_Boltzmann``         Stefan-Boltzmann constant :math:`\sigma`
    ``Wien``                     Wien displacement law constant
    ``Rydberg``                  Rydberg constant
    ``m_e``                      electron mass
    ``electron_mass``            electron mass
    ``m_p``                      proton mass
    ``proton_mass``              proton mass
    ``m_n``                      neutron mass
```

```
print(constants.liter)
```

```
0.001
```

```
from scipy import cluster
help(cluster)
```

```
Help on package scipy.cluster in scipy:

NAME
    scipy.cluster

DESCRIPTION
    =========================================
    Clustering package (:mod:`scipy.cluster`)
    =========================================

    .. currentmodule:: scipy.cluster

    .. toctree::
       :hidden:

       cluster.vq
       cluster.hierarchy

    Clustering algorithms are useful in information theory, target detection,
    communications, compression, and other areas. The `vq` module only
    supports vector quantization and the k-means algorithms.

    The `hierarchy` module provides functions for hierarchical and
    agglomerative clustering.  Its features include generating hierarchical
    clusters from distance matrices,
    calculating statistics on clusters, cutting linkages
    to generate flat clusters, and visualizing clusters with dendrograms.

PACKAGE CONTENTS
    _hierarchy
    _optimal_leaf_ordering
    _vq
    hierarchy
    tests (package)
    vq

DATA
    __all__ = ['vq', 'hierarchy']

FILE
    /usr/local/lib/python3.10/dist-packages/scipy/cluster/__init__.py
```

```
from scipy import special
a=special.exp10(3)
print(a)
```

```
1000.0
```

```
b= special.exp2(3)
print(b)
```

```
8.0
```

```
c=special.sindg(0)
print(c)
```

```
0.0
```

```
d=special.cosdg(90)
print(d)
```

```
-0.0
```

```
from scipy import integrate
help(integrate)
```

```
Help on package scipy.integrate in scipy:

NAME
    scipy.integrate

DESCRIPTION
    =============================================
    Integration and ODEs (:mod:`scipy.integrate`)
    =============================================

    .. currentmodule:: scipy.integrate

    Integrating functions, given function object
    ============================================

    .. autosummary::
       :toctree: generated/

       quad          -- General purpose integration
       quad_vec      -- General purpose integration of vector-valued functions
       dblquad       -- General purpose double integration
       tplquad       -- General purpose triple integration
       nquad         -- General purpose N-D integration
       fixed_quad    -- Integrate func(x) using Gaussian quadrature of order n
       quadrature    -- Integrate with given tolerance using Gaussian quadrature
       romberg       -- Integrate func using Romberg integration
       newton_cotes  -- Weights and error coefficient for Newton-Cotes integration
       qmc_quad      -- N-D integration using Quasi-Monte Carlo quadrature
       IntegrationWarning -- Warning on issues during integration
       AccuracyWarning  -- Warning on issues during quadrature integration

    Integrating functions, given fixed samples
    ==========================================

    .. autosummary::
       :toctree: generated/

       trapezoid            -- Use trapezoidal rule to compute integral.
       cumulative_trapezoid -- Use trapezoidal rule to cumulatively compute integral.
       simpson              -- Use Simpson's rule to compute integral from samples.
       romb                 -- Use Romberg Integration to compute integral from
                            -- (2**k + 1) evenly-spaced samples.

    .. seealso::

       :mod:`scipy.special` for orthogonal polynomials (special) for Gaussian
       quadrature roots and weights for other weighting factors and regions.

    Solving initial value problems for ODE systems
    ==============================================

    The solvers are implemented as individual classes, which can be used directly
```

```
    (low-level usage) or through a convenience function.

    .. autosummary::
       :toctree: generated/
```

```
i=scipy.integrate.quad(lambda x :special.exp10(x),0,1)
print(i)
```

```
(3.9086503371292665, 4.3394735994897923e-14)
```

```
e=lambda x,y:8*x+6*y
f=lambda x:0
g= lambda x:1
integrate.dblquad(e,0,2,f,g)
```

```
(20.0, 2.220446049250313e-13)
```

```
from scipy import linalg
a=np.array([[1,2],[3,4]])
b=linalg.inv(a)
print(b)
```

```
[[-2.   1. ]
 [ 1.5 -0.5]]
```

```
from ast import increment_lineno
%matplotlib inline
import matplotlib.pyplot as plt
from scipy import interpolate
x=np.arange(5,20)
y=np.exp(x/3.0)
f=interpolate.interp1d(x,y)
x1=np.arange(6,12)
y1=f(x1)
plt.plot(x,y,'o',x1,y1,'--')
plt.show()
```