

Python for Data Analysis book

Chapter 4

Introduction to Numpy

-- Then we'll talk about your exam!

Numpy: Numerical Python

Numpy is a core library built and used as the backbone of most high level scientific and numerical analysis packages.

Numpy provides data structures and functionality

Applications on

info450_fall_2020



Cha



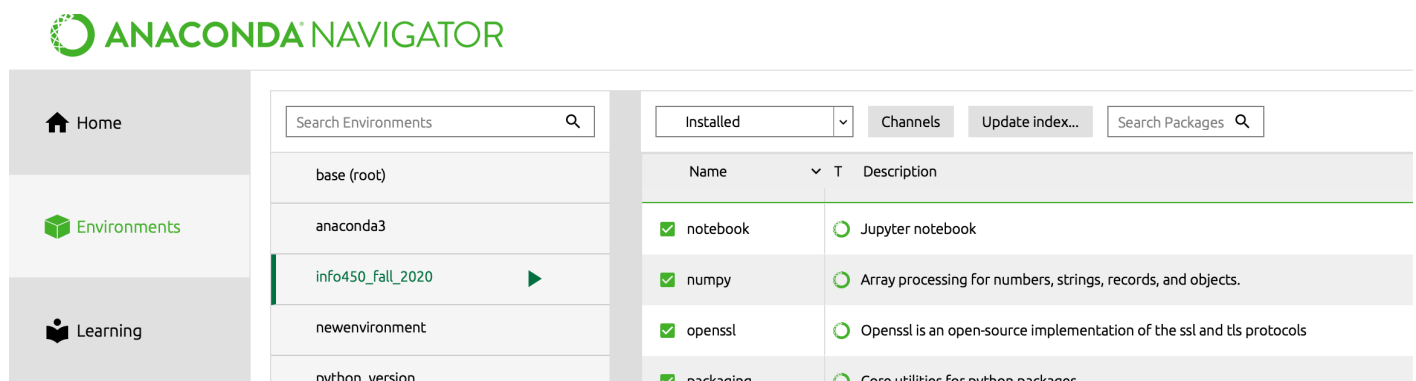
Notebook

[↗ 6.1.4](#)

Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.

Streamline
developm
task r

Launch



ndarray: an efficient multi-dimensional

array providing fast array-oriented arithmetic operations and flexible broadcasting capabilities.

Mathematical functions for fast operations on entire arrays of data without having to write loops

Tools for reading/writing array data to disk and working with memory-mapped files

Linear Algebra, random number generation, Fourier transform capabilities

A C API for connecting NumPy with libraries written in C, C++ and FORTRAN

Numpy stores data efficiently by keeping it in contiguous blocks of memory, independent of other Python objects

```
In [9]: # Import the numpy library, aliasing it to np.. Not sure why we always  
do that  
import numpy as np
```

```
In [10]: data = np.random.randn(2,3)
```

```
In [11]: data
```

```
Out[11]: array([[ -2.13048994, -0.93506454,  0.07414567],  
               [ 0.03039112,  1.68584412,  1.89978145]])
```

```
In [12]: # multiply each item in the array by 10  
data * 10
```

```
Out[12]: array([[ -21.30489941,  -9.35064543,   0.74145669],  
               [  0.30391115,  16.85844124,  18.99781448]])
```

```
In [13]: # Add data to data (adding two multi dimension arrays)  
data + data
```

```
Out[13]: array([[ -4.26097988, -1.87012909,  0.14829134],  
               [ 0.06078223,  3.37168825,  3.7995629 ]])
```

```
In [14]: # size of dimensions  
data.shape
```

```
Out[14]: (2, 3)
```

```
In [15]: # Data type stored in array  
data.dtype
```

```
Out[15]: dtype('float64')
```

From the book, they refer to a 'ndarray' object multiple ways:

1. Array
2. NumPy array
3. ndarray

```
In [17]: #Create arrays  
list_data1 = [6, 7.5, 8, 0, 1]  
ndarray_data1 = np.array(list_data1)
```

```
In [18]: ndarray_data1
```

```
Out[18]: array([6. , 7.5, 8. , 0. , 1. ])
```

```
In [19]: ndarray_data1.shape
```

```
Out[19]: (5,)
```

```
In [20]: ndarray_data1.dtype
```

```
Out[20]: dtype('float64')
```

```
In [21]: # Create a multi dimensional array  
list_data2 = [[1, 2, 3, 4], [5, 6, 7, 8]]  
ndarray_data2 = np.array(list_data2)
```

```
In [22]: ndarray_data2
```

```
Out[22]: array([[1, 2, 3, 4],  
               [5, 6, 7, 8]])
```

```
In [23]: ndarray_data2.dtype
```

```
Out[23]: dtype('int64')
```

```
In [24]: ndarray_data2.ndim
```

```
Out[24]: 2
```

```
In [25]: ndarray_data2.shape
```

```
Out[25]: (2, 4)
```

dtype is an 'inferred' meta data attribute of the array

numpy tries to guess based on the data in the array

Numpy has a suite of functions for creating new arrays

```
In [28]: # Create an array with a lot of zeros  
np.zeros(10)
```

```
Out[28]: array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

```
In [29]: np.zeros((3, 6))
```

```
Out[29]: array([[0., 0., 0., 0., 0., 0.],  
               [0., 0., 0., 0., 0., 0.],  
               [0., 0., 0., 0., 0., 0.]])
```

```
In [30]: np.empty((2,3,2))
```

```
Out[30]: array([[[0.00000000e+000, 0.00000000e+000],  
                [0.00000000e+000, 0.00000000e+000],  
                [0.00000000e+000, 0.00000000e+000]],  
               [[0.00000000e+000, 0.00000000e+000],  
                [0.00000000e+000, 0.00000000e+000],  
                [0.00000000e+000, 5.24862813e+170]])
```

**** WARNING ***

It's not safe to assume that `np.empty` will return an array of all zeros. In some cases, it may return uninitialized 'garbage' values. Uninitialized garbage means: Whatever was already in memory.

```
In [32]: # arange:  numpy , array value of the built in range function  
for x in range(10):  
    print(x)
```

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

```
In [33]: np.arange(15)
```

```
Out[33]: array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14])
```

- `array`
- `asarray`
- `arange`
- `ones`,
- `ones_like` # duplicates the structure/shape of an existing array
- `zeros`,
- `zeros_like`
- `empty`,
- `empty_like`
- `full`,
- `full_like`
- `eye`, `identity` # Create a square $N \times N$ identity matrix (1s on the diagonal, 0s elsewhere)

```
In [35]: np.eye(12)
```

```
Out[35]: array([[1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
 [0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
 [0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
 [0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0.],
 [0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0.],
 [0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0.],
 [0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0.],
 [0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0.],
 [0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0.],
 [0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0.],
 [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0.],
 [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1.]])
```

```
In [36]: np.identity(12)
```

```
Out[36]: array([[1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
 [0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
 [0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
 [0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0.],
 [0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0.],
 [0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0.],
 [0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0.],
 [0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0.],
 [0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0.],
 [0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0.],
 [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0.],
 [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1.]])
```

```
In [37]: """
You can explicitly tell an ndarray what data type to have inside of it.
done with a parameter for the different functions
"""
np.array([1,2,3], dtype=np.float64)
```

```
Out[37]: array([1., 2., 3.])
```

```
In [38]: np.array([1,2,3], dtype=np.float64).dtype
```

```
Out[38]: dtype('float64')
```

```
In [39]: np.array([1,2,3], dtype=np.int32)
```

```
Out[39]: array([1, 2, 3], dtype=int32)
```

```
In [40]: np.array([1,2,3], dtype=np.int32).dtype
```

```
Out[40]: dtype('int32')
```

Numpy supports a wide range of data types, but you don't have to memorize each and everyone

- int (8/16/32/64)
- float (16, 32, 64, 128)
- complex (64, 128, 256) - complex type represented by two numbers
- bool
- object
- string_ (fixed size ascii, S abbreviation, S10)
- unicode_ (fixed size Unicode, U abbreviation, U12)

```
In [42]: """  
         We can explicitly cast/convert some dtypes to others  
         """  
         int_arr = np.array([1,2,3,4])  
         int_arr
```

```
Out[42]: array([1, 2, 3, 4])
```

```
In [43]: int_arr.dtype
```

```
Out[43]: dtype('int64')
```

```
In [44]: float_arr = int_arr.astype(np.float64)  
         float_arr
```

```
Out[44]: array([1., 2., 3., 4.])
```

```
In [45]: float_arr.dtype
```

```
Out[45]: dtype('float64')
```

I'm skipping a bunch of information on 'slicing' the array and other utilities like 'resizing'

Please read chapter 4.1 in your book. That material CAN be used in a quiz.

Fun functions for arrays

UNARY Functions

Functions that act on one ndarray

- exp - calculate the exponential of each number
- sqrt - square root
- exp2 - $2^{**}x$
- abs, fabs
- square
- log, log10, log2, log1p
- sign
- ceil
- floor
- rint
- modf
- isnan
- isfinite, isinf
- cos, cosh, sin, sinh, tan, tanh
- arccos, archosh, arcsin, arcsinh, arctan, arctanh
- logical_not

```
In [48]: int_arr2 = np.arange(12, dtype=np.int32)
         int_arr2
```

```
Out[48]: array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11], dtype=int32)
```

```
In [49]: np.exp2(int_arr2)
```

```
Out[49]: array([1.000e+00, 2.000e+00, 4.000e+00, 8.000e+00, 1.600e+01, 3.200e+01,
               6.400e+01, 1.280e+02, 2.560e+02, 5.120e+02, 1.024e+03, 2.048e+03])
```

```
In [50]: np.exp2(int_arr2).dtype
```

```
Out[50]: dtype('float64')
```

```
In [51]: np.exp2(int_arr2).astype(np.int32)
```

```
Out[51]: array([  1,   2,   4,   8,  16,  32,  64, 128, 256, 512, 1024,
               2048], dtype=int32)
```

BINARY Functions

Functions that act on TWO ndarrays

- add
- subtract
- multiply
- divide, floor_divide
- power
- maximum, fmax - (fmax ignore NaN)
- minimum, fmin - (fmin ignores NaN)
- mod - modulus
- copysign - copies sign of second argument to first
- greater, greater_equal, less, less_equal, equal, not_equal
- logical_and, logical_or, logical_xor

```
In [53]: first_arr = np.array([1,2,3,4])  
second_arr = np.array([1,2,3,4])  
np.add(first_arr, second_arr)
```

```
Out[53]: array([2, 4, 6, 8])
```

```
In [54]: first_arr = np.array([1,2,3,4])  
second_arr = np.array([10,20,30,40])  
np.subtract(first_arr, second_arr)
```

```
Out[54]: array([ -9, -18, -27, -36])
```

```
In [55]: np.subtract(second_arr, first_arr)
```

```
Out[55]: array([ 9, 18, 27, 36])
```

```
In [56]: first_arr
```

```
Out[56]: array([1, 2, 3, 4])
```

```
In [57]: second_arr
```

```
Out[57]: array([10, 20, 30, 40])
```

```
In [58]: third_arr = np.subtract(second_arr, first_arr)
```

```
In [59]: third_arr
```

```
Out[59]: array([ 9, 18, 27, 36])
```

Fun visualization, if we get this far.

Installed

Channels

Update index...

matplot

Name	T	Description
✓ matplotlib		Publication quality figures in python
✓ matplotlib-base		

```
In [60]: import matplotlib.pyplot as plt
points = np.arange(-5, 5, 0.01)
xs, ys = np.meshgrid(points, points)
```

```
In [62]: xs
```

```
Out[62]: array([[ -5.   , -4.99, -4.98, ...,  4.97,  4.98,  4.99],
                [ -5.   , -4.99, -4.98, ...,  4.97,  4.98,  4.99],
                [ -5.   , -4.99, -4.98, ...,  4.97,  4.98,  4.99],
                ...,
                [ -5.   , -4.99, -4.98, ...,  4.97,  4.98,  4.99],
                [ -5.   , -4.99, -4.98, ...,  4.97,  4.98,  4.99],
                [ -5.   , -4.99, -4.98, ...,  4.97,  4.98,  4.99]])
```

```
In [63]: ys
```

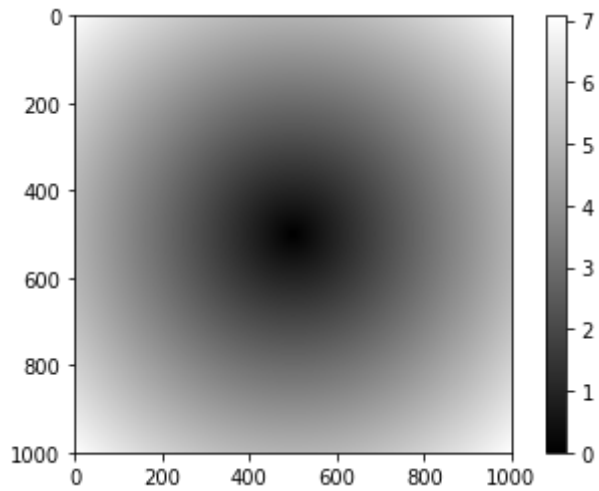
```
Out[63]: array([[ -5.   , -5.   , -5.   , ..., -5.   , -5.   , -5.   ],
                [-4.99, -4.99, -4.99, ..., -4.99, -4.99, -4.99],
                [-4.98, -4.98, -4.98, ..., -4.98, -4.98, -4.98],
                ...,
                [ 4.97,  4.97,  4.97, ...,  4.97,  4.97,  4.97],
                [ 4.98,  4.98,  4.98, ...,  4.98,  4.98,  4.98],
                [ 4.99,  4.99,  4.99, ...,  4.99,  4.99,  4.99]])
```

```
In [66]: z = np.sqrt(xs ** 2 + ys **2)
```

z

```
In [70]: plt.imshow(z, cmap=plt.cm.gray)
plt.colorbar()
```

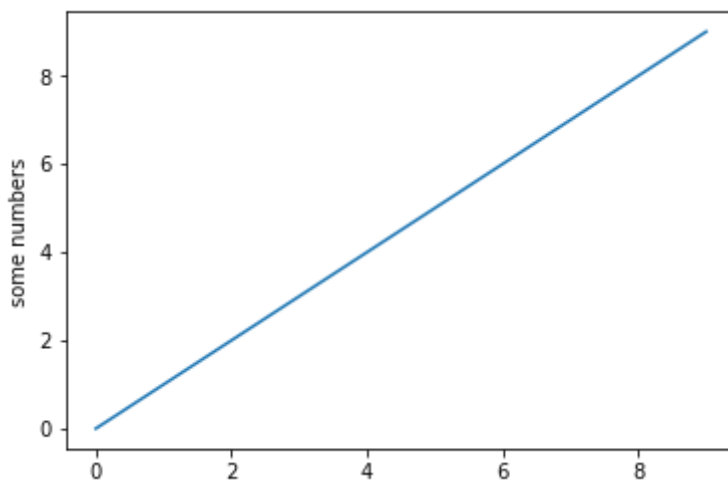
Out[70]: <matplotlib.colorbar.Colorbar at 0x7fc2426ffa90>



```
In [83]: new_xs = np.arange(10)
new_xs
```

Out[83]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

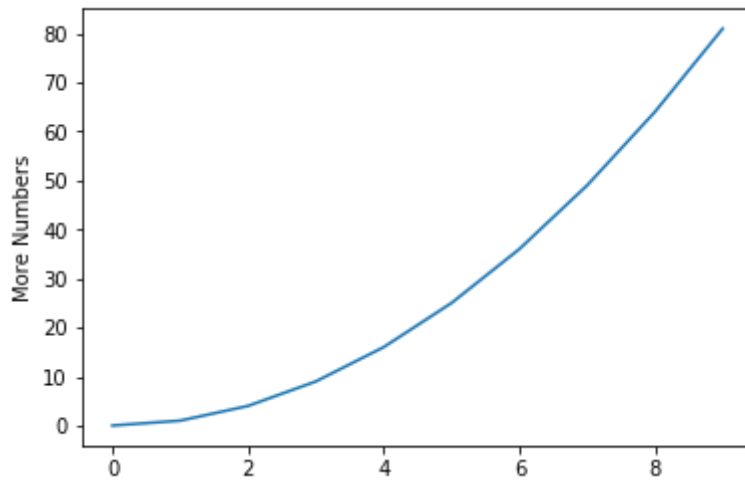
```
In [84]: plt.plot(new_xs)
plt.ylabel('some numbers')
plt.show()
```



```
In [87]: np.power(new_xs, 2)
```

Out[87]: array([0, 1, 4, 9, 16, 25, 36, 49, 64, 81])

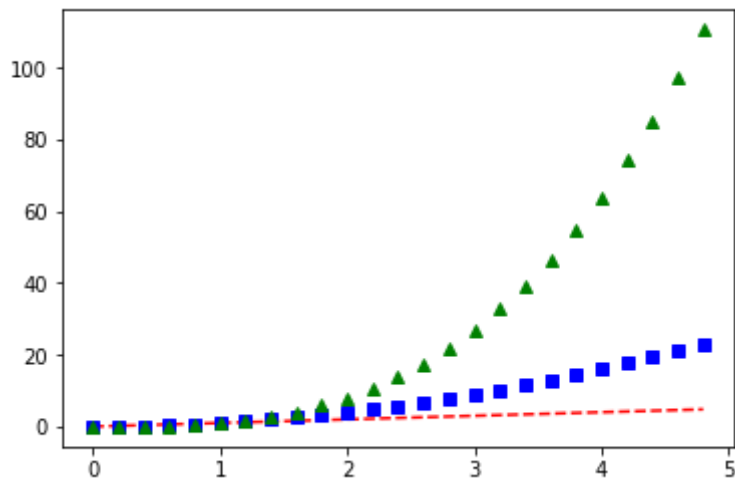
```
In [88]: plt.plot(np.power(new_xs, 2))
plt.ylabel("More Numbers")
plt.show()
```



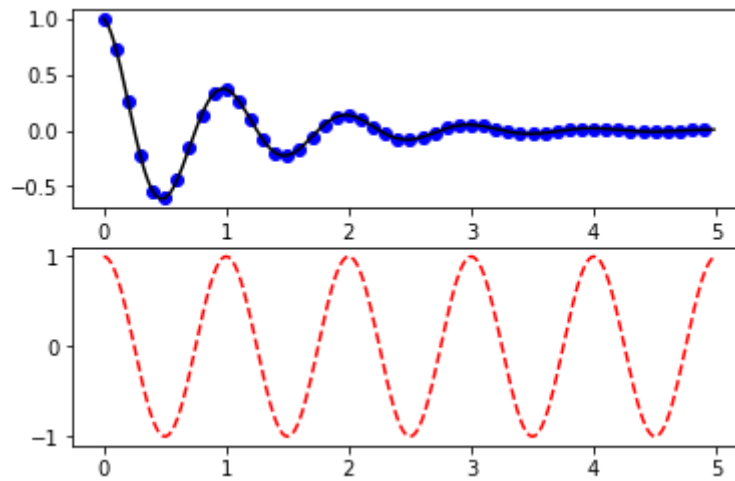
```
In [89]: t = np.arange(0., 5., 0.2)

# red dashes, blue squares and green triangles
"""
t, t, "r--" # x=t, y=t, red dashes
t, t**2, "bs" # x = t, y = t squared, blue square
t, t**3, g^ # x = t, y = t ^3, green triangles]
"""

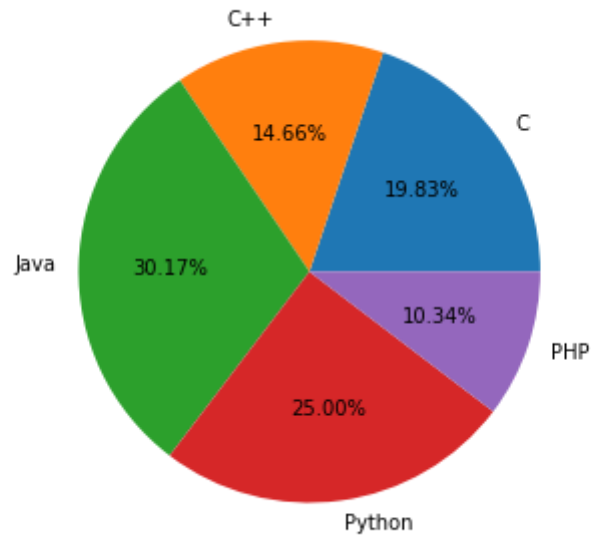
plt.plot(t, t, 'r--', t, t**2, 'bs', t, t**3, 'g^')
plt.show()
```



```
In [90]: def f(t):  
         return np.exp(-t) * np.cos(2*np.pi*t)  
  
t1 = np.arange(0.0, 5.0, 0.1)  
t2 = np.arange(0.0, 5.0, 0.02)  
  
plt.figure()  
plt.subplot(211)  
plt.plot(t1, f(t1), 'bo', t2, f(t2), 'k')  
  
plt.subplot(212)  
plt.plot(t2, np.cos(2*np.pi*t2), 'r--')  
plt.show()
```



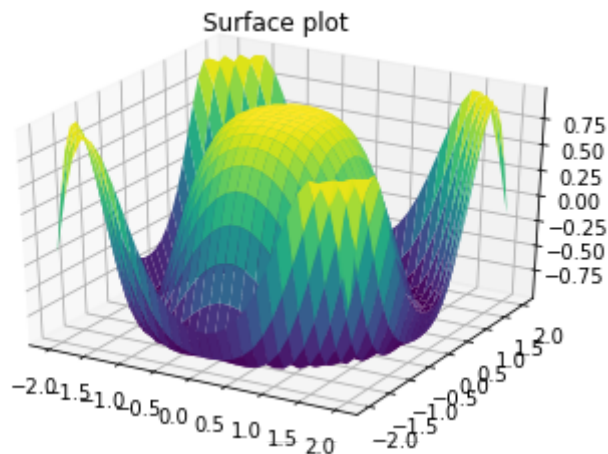
```
In [94]: from matplotlib import pyplot as plt
import numpy as np
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
ax.axis('equal')
langs = ['C', 'C++', 'Java', 'Python', 'PHP']
students = [23,17,35,29,12]
ax.pie(students, labels = langs, autopct='%1.2f%%')
plt.show()
```



```
In [95]: from mpl_toolkits import mplot3d
import numpy as np
import matplotlib.pyplot as plt
x = np.outer(np.linspace(-2, 2, 30), np.ones(30))
y = x.copy().T # transpose
z = np.cos(x ** 2 + y ** 2)

fig = plt.figure()
ax = plt.axes(projection='3d')

ax.plot_surface(x, y, z, cmap='viridis', edgecolor='none')
ax.set_title('Surface plot')
plt.show()
```



https://www.tutorialspoint.com/matplotlib/matplotlib_3d_contour_plot.htm
[\(https://www.tutorialspoint.com/matplotlib/matplotlib_3d_contour_plot.htm\)](https://www.tutorialspoint.com/matplotlib/matplotlib_3d_contour_plot.htm)