Matplotlib pyplot

**Title,xlabel.ylabel,xtick,ytick,annotation,legend,fontsize,xlim,ylim**

**import matplotlib.pyplot as plt  
import numpy as np  
  
small\_size=8  
medium\_size=12  
large\_size=16**  
x=(1,2,3,4,5,6,7)  
y=(3,6,9,12,16,18,20)  
y1=(2,4,6,8,10,12,14)  
y2=(4,8,12,16,20,24,28)  
  
plt.xlabel("Days")  
plt.ylabel("Prices")  
plt.rc("font",size=small\_size)  
plt.rc("axes",titlesize=medium\_size)  
plt.rc("axes",labelsize=small\_size)  
plt.rc("figure",titlesize=large\_size)  
plt.plot(x,y,"blue",label="Tomato")  
plt.axis([0,8,0,30])  
plt.annotate("Tomato",xy=(2,6),xytext=(2.5,6),arrowprops=dict(facecolor="black",shrink=0.05))

plt.plot(x,y1,"red",label="Potato")  
plt.plot(x,y2,"green",label="Brinjal")  
plt.title("Comparison of prices")  
plt.legend()  
plt.show()

plt.rc('font', size=SMALL\_SIZE) # controls default text sizes

plt.rc('axes', titlesize=SMALL\_SIZE) # fontsize of the axes title

plt.rc('axes', labelsize=MEDIUM\_SIZE) # fontsize of the x and y labels

plt.rc('xtick', labelsize=SMALL\_SIZE) # fontsize of the tick labels

plt.rc('ytick', labelsize=SMALL\_SIZE) # fontsize of the tick labels

plt.rc('legend', fontsize=SMALL\_SIZE) # legend fontsize

plt.rc('figure', titlesize=BIGGER\_SIZE) # fontsize of the figure title

pie plots

## Creating Pie Charts

With Pyplot, you can use the pie() function to draw pie charts:

As you can see the pie chart draws one piece (called a wedge) for each value in the array (in this case [35, 25, 25, 15]).

By default the plotting of the first wedge starts from the x-axis and move *counterclockwise*:

As mentioned the default start angle is at the x-axis, but you can change the start angle by specifying a startangle parameter.

Maybe you want one of the wedges to stand out? The explode parameter allows you to do that.

import matplotlib.pyplot as plt  
import numpy as np  
  
y = np.array([35, 25, 25, 15])  
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]  
myexplode = [0.2, 0, 0, 0]

plt.pie(y, labels = mylabels, startangle = 90, explode = myexplode)  
plt.show()

Polar plots

Rendering in local webserver

Plotly – interacrtive library

import plotly.express as px  
import plotly.graph\_objects as go  
import pandas as pd  
import plotly.io as pio  
pio.renderers  
  
df = pd.DataFrame(dict(  
 r=[1, 5, 2, 2, 3],  
 theta=['processing cost','mechanical properties','chemical stability',  
 'thermal stability', 'device integration']))  
fig = px.line\_polar(df, r='r', theta='theta', line\_close=True)  
  
  
#fig = go.Figure(  
# data=[go.Bar(y=[2, 1, 3])],  
# layout\_title\_text="A Figure Displaying Itself"  
#)  
fig.show(renderer="browser")

in browser, if the plot is rendered, it is not interactive

import numpy as np  
import matplotlib.pyplot as plt  
  
  
categories = ['Food Quality', 'Food Variety', 'Service Quality', 'Ambiance', 'Affordability']  
  
restaurant\_1 = [4, 4, 5, 4, 3]  
restaurant\_2 = [5, 5, 4, 5, 2]  
restaurant\_3 = [3, 4, 5, 3, 5]  
  
#label\_loc = np.linspace(start=0, stop=2 \* np.pi, num=len(restaurant\_1))  
label\_loc = [1,2,3,4,5]  
print(label\_loc)  
plt.figure(figsize=(8, 8))  
plt.subplot(polar=True)  
plt.plot(label\_loc, restaurant\_1, label='Restaurant 1')  
plt.plot(label\_loc, restaurant\_2, label='Restaurant 2')  
plt.plot(label\_loc, restaurant\_3, label='Restaurant 3')  
plt.title('Restaurant comparison', size=20)  
lines, labels = plt.thetagrids(np.degrees(label\_loc), labels=categories)  
plt.legend()  
plt.show()

**Table**

import matplotlib.pyplot as plt  
import pandas as pd  
x=[ i for i in range(10)]  
y=["{:02X}".format(i\*2) for i in range(10)]  
z=[[i for i in range(10)] for j in range(10)]  
print(x)  
print(y)  
print(z)  
  
fig, ax = plt.subplots()  
ax.set\_axis\_off()  
plt.table(cellText=z,rowLabels=y,colLabels=x,rowColours=["Palegreen"]\*10,colColours=["Palegreen"]\*10,cellLoc="center",loc="upper left")  
plt.show()

**Coloring the cells**

import matplotlib.pyplot as plt  
import pandas as pd  
import numpy as np  
x=[ i for i in range(10)]  
y=["{:02X}".format(i\*2) for i in range(3)]  
z=[[i for i in range(10)] for j in range(3)]  
print(x)  
print(y)  
print(z)  
  
fig, ax = plt.subplots()  
ax.set\_axis\_off()  
rcol=plt.cm.BuPu(np.full(len(y),0.1))  
ccolor=[["r","r","r","r","r","r","r","r","r","r"],["r","r","r","r","r","r","r","r","r","r"],["r","r","r","r","r","r","r","r","r","r"]]  
plt.table(cellText=z,cellColours=ccolor,rowLabels=y,colLabels=x,rowColours=rcol,colColours=["Palegreen"]\*10,cellLoc="center",loc="upper left")  
plt.show()

matplotlib.pyplot.table(cellText=None, cellColours=None, cellLoc='right', colWidths=None, rowLabels=None, rowColours=None, rowLoc='left', colLabels=None, colColours=None, colLoc='center', loc='bottom', bbox=None, edges='closed', \*\*kwargs)

Specifying one of **cellText or cellColours** as a parameter to the matplotlib table function is mandatory. These parameters must be **2D lists**, in which the outer lists define the rows and the **inner list** define the column values of each row. Any other dimensional data will not be accepted by the function.

The table can optionally have row and column headers. **rowLabels, rowColours, rowLoc, and colLabels, colColours, colLoc** respectively are optional parameters for the function. **\*\*kwargs** are optional parameters to adjust for the size of the 2-d list passed as a parameter.

**Parameters:**

* **cellText**: The texts to place into the table cells.
* **cellColours**: The [background colors](https://www.pythonpool.com/matplotlib-background-color/) of the cells.
* **cellLoc:** The alignment of the text within the cells. (default: ‘right’)
* **colWidths (optional):**The column widths in units of the axes.
* **rowLabels (optional):**The text of the row header cells.
* **rowColours (optional):**The colors of the row header cells.
* **rowLoc (optional):** The text alignment of the row header cells. (default: ‘left’)
* **colLabels (optional):** The text of the column header cells.
* **colColours (optional):** The colors of the column header cells.
* **colLoc (optional):** The text alignment of the column header cells.(default: ‘left’)
* **Loc (optional):**This parameter is the position of the cell with respect to ax.
* **bbox (optional):**This parameter is the bounding box to draw the table into.
* **edges (optional):**This parameter is the cell edges to be drawn with a line.
* **\*\*kwargs**: Used to control table properties.

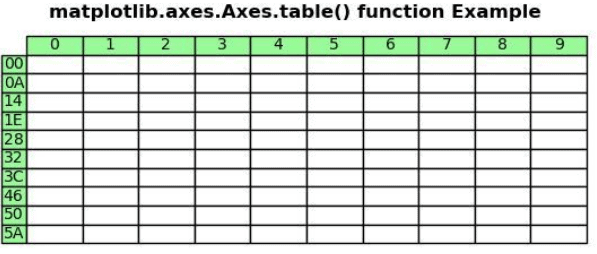
**Return type :**

The matplotlib.pylot.table() method returns the table created passing the required data as parameters. This table object can be grabbed to change the specific values within the table. This object refers to the [matplotlib.table.Table()](https://matplotlib.org/3.1.1/api/table_api.html#matplotlib.table.Table) object. The table consists of 2d grid that can be index by using rows and columns. Moreover, you can also change the font family of the table.

**Implementation of Matplotlib table**

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20  21 | import matplotlib.pyplot as plt    val1 = ["{:X}".format(i) for i in range(10)]  val2 = ["{:02X}".format(10 \* i) for i in range(10)]  val3 = [["" for c in range(10)] for r in range(10)]    fig, ax = plt.subplots()  ax.set\_axis\_off()  table = ax.table(      cellText = val3,      rowLabels = val2,      colLabels = val1,      rowColours =["palegreen"] \* 10,      colColours =["palegreen"] \* 10,      cellLoc ='center',      loc ='upper left')    ax.set\_title('matplotlib.axes.Axes.table() function Example',               fontweight ="bold")    plt.show() |

**Output:**



**Explanation:**

In the above example, matplotlib.pyplot.table() accepts the 2-d list as the parameter. The parameters passed are as follows:- cellText = val3, rowLabels = val2, colLabels = val1, rowColours =[“palegreen”] \* 10, colColours =[“palegreen”] \* 10, cellLoc =’center’, loc =’upper left’. Val1, val2 and val3 runs for loops each to generate values for column lables, row lables and cell [text](https://www.pythonpool.com/matplotlib-text/), respectively. Setting rowColours and colColours to ‘palegreen’ changes the color of row and column header cells to palegreen. Setting cellLoc =’center’ sets the alignment of header [row](https://en.wikipedia.org/wiki/Row-_and_column-major_order#:~:text=In%20computing%2C%20row%2Dmajor%20order,such%20as%20random%20access%20memory.&text=Data%20layout%20is%20critical%20for,written%20in%20different%20programming%20languages.) and column values to center. And finally, loc =’upper left’ sets the alignment of the Title to upper left.

Colormaps

import numpy as np  
import matplotlib.pyplot as plt  
import matplotlib.colors as clr  
  
x=np.linspace(-3,3,20)  
y=np.linspace(-5,5,20)  
z=np.linspace(-1,1,20)  
print(z)  
  
figure, (axes1, axes2) = plt.subplots(ncols=2)  
axes1.set\_title("Default Color Assignment")  
colors1=["green","orange","red","blue"]  
cmap=clr.ListedColormap(colors1)  
  
axes1.scatter(x,y,c=z)  
axes2.set\_title("Userdefined colormaps")  
scatter=axes2.scatter(x,y,c=z,cmap=cmap)  
figure.colorbar(scatter)  
plt.show()

Explanation

Here, x,y,z – three attributes are used.

For z attribute values colors are assigned.

ListedColormap function generate colormap for the colors list

Two subplots are created

First subplots to show the pyplot default colors

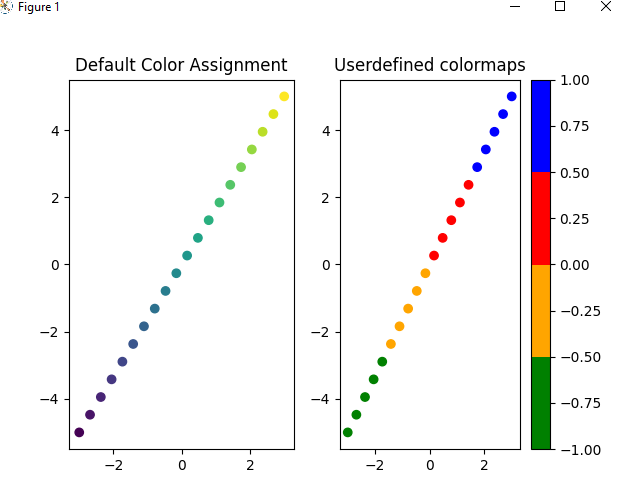
Second subplot to show the scalar data presented with user designed colormaps

Color legend is included using colorbar

To create random numbers from normal function z=np.random.randn(1,50)

To create co-ordiante matrix using the array vectors a,b

A , B = np.meshgrid( a,b)



Contours

import matplotlib.pyplot as plt  
import numpy as np  
import matplotlib.colors  
  
a = np.linspace(-3, 3)  
#print(a)  
A, B = np.meshgrid(a, a)  
print(A)  
X = np.exp(-(A \*\* 2 + B \*\* 2))  
figure, (axes1, axes2) = plt.subplots(ncols=2)  
  
colors = ["green", "orange",  
"gold", "blue", "k",  
"#550011", "purple",  
"red"]  
  
axes1.set\_title(" color list")  
contour = axes1.contourf(A, B, X,  
colors = colors)  
  
axes2.set\_title("with colormap")  
cmap = matplotlib.colors.ListedColormap(colors)  
contour = axes2.contourf(A, B, X, cmap=cmap)  
figure.colorbar(contour)  
  
plt.show()

X= 1,2,3,4,5

Y=7,9,10,12,15

Z=9,10,10,8,6,7

Z1=9,0,0,0,0,0,10,0,0,0

Np.Meshgrid(X,Y)

(1,7) (1,9) (1,10)(1,12)(1,15)

(2,7)(2,9)(2,10)(2,12)(2,15)

(3,7),(3,9)(3,10)(3,12)(3,15)

X=1,2,3,4

Y=5,6,7,8

[X,Y]=[[(1,5),(1,6,(1,7),(1,8)],[(2,5),(2,6),(2,7),(2,8)],[(3,5),(3,6),(3,7),(3,8)],[(4,5),(4,6),(4,7),(4,8)]

[[1,0,0,0][0,1,0,0].[0,0,1,0],[0,0,0,1]]

Explanation:

Here three attributes A,B,X are used

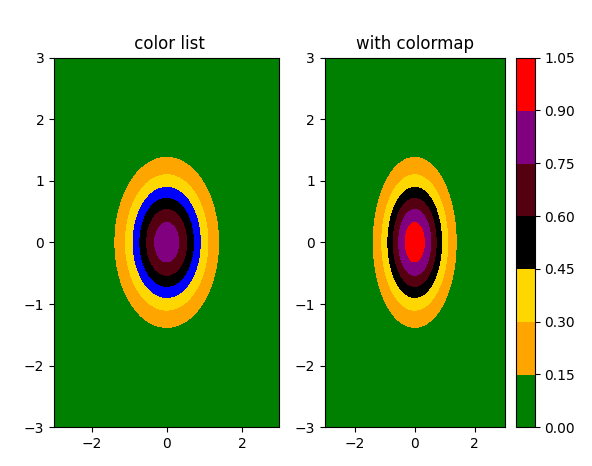
For X attribute value colormap is used.

Contours are constructed based similar X values

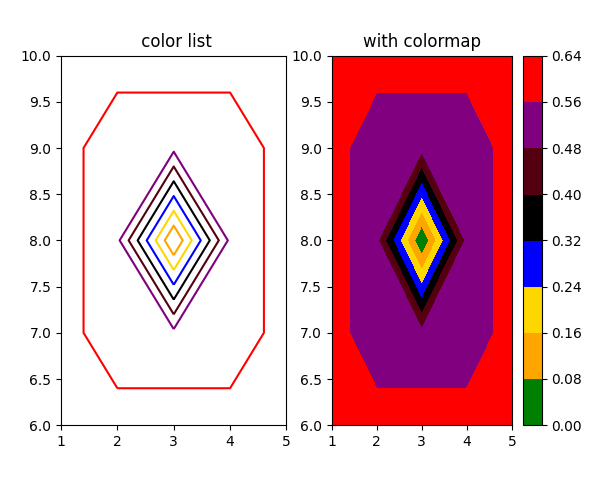
Two subplots are presented

First subplot assigns default pyplot colors to X values

Second subplot assigns user defined colormap values



import matplotlib.pyplot as plt  
import numpy as np  
import matplotlib.colors  
  
a = [1,2,3,4,5]  
b = [6,7,8,9,10]  
#print(a)  
A, B = np.meshgrid(a, b)  
print(A)  
print(B)  
#X = np.exp(-(A \*\* 2 + B \*\* 2))  
X =[[0.6,0.6,0.6,0.6,0.6], [0.6,.5,0.5,0.5,0.6],[0.6,0.5,0,0.5,0.6] , [0.6,0.5,0.5,0.5,0.6],[0.6,0.6,0.6,0.6,0.6]]  
print(X)  
  
  
figure, (axes1, axes2) = plt.subplots(ncols=2)  
  
colors = ["green", "orange",  
"gold", "blue", "k",  
"#550011", "purple",  
"red"]  
cmap = matplotlib.colors.ListedColormap(colors)  
axes1.set\_title(" color list")  
contour = axes1.contour(A, B, X,  
cmap = cmap)  
  
axes2.set\_title("with colormap")  
  
contour = axes2.contourf(A, B, X, cmap=cmap)  
figure.colorbar(contour)  
  
plt.show()



Python Markers(symbol), size, colors(hue)

“s” – square, “d” – diomand, ‘o’ – circle, ‘^’ – trianle

S=10 -> to specify marker size

import matplotlib.pyplot as plt  
import numpy as np  
  
x=np.linspace(1,30,30)  
y=np.random.randint(10000,size=(30))  
z=np.random.exponential(15000,size=(30))  
  
fig,(axes1,axes2)=plt.subplots(1,2)  
axes1.set\_title("Tamilnadu")  
axes1.set\_xlabel("Day")  
axes1.set\_ylabel("Daily Infection")  
#axes1.plot(x,y)  
#axes2.plot(x,z)  
#axes1.plot(x,y,marker='s',markersize=10,c="red")  
#axes2.plot(x,z,marker='d',markersize=20,c="blue")  
axes1.scatter(x,y,marker='o',s=10,c="red")  
axes1.legend(loc=8,framealpha=1, fontsize=8)  
axes2.scatter(x,z,marker='^',s=20,c="blue")  
axes2.set\_title("Kerala")  
axes2.set\_xlabel("Day")  
axes2.set\_ylabel("Daily Infection")  
plt.show()

Matplotlib.markers

| **marker** | **symbol** | **description** |
| --- | --- | --- |
| "." | m00 | point |
| "," | m01 | pixel |
| "o" | m02 | circle |
| "v" | m03 | triangle\_down |
| "^" | m04 | triangle\_up |
| "<" | m05 | triangle\_left |
| ">" | m06 | triangle\_right |
| "1" | m07 | tri\_down |
| "2" | m08 | tri\_up |
| "3" | m09 | tri\_left |
| "4" | m10 | tri\_right |
| "8" | m11 | octagon |
| "s" | m12 | square |
| "p" | m13 | pentagon |
| "P" | m23 | plus (filled) |
| "\*" | m14 | star |
| "h" | m15 | hexagon1 |
| "H" | m16 | hexagon2 |
| "+" | m17 | plus |
| "x" | m18 | x |
| "X" | m24 | x (filled) |
| "D" | m19 | diamond |
| "d" | m20 | thin\_diamond |
| "|" | m21 | vline |
| "\_" | m22 | hline |
| 0 (TICKLEFT) | m25 | tickleft |
| 1 (TICKRIGHT) | m26 | tickright |
| 2 (TICKUP) | m27 | tickup |
| 3 (TICKDOWN) | m28 | tickdown |
| 4 (CARETLEFT) | m29 | caretleft |
| 5 (CARETRIGHT) | m30 | caretright |
| 6 (CARETUP) | m31 | caretup |
| 7 (CARETDOWN) | m32 | caretdown |
| 8 (CARETLEFTBASE) | m33 | caretleft (centered at base) |
| 9 (CARETRIGHTBASE) | m34 | caretright (centered at base) |
| 10 (CARETUPBASE) | m35 | caretup (centered at base) |
| 11 (CARETDOWNBASE) | m36 | caretdown (centered at base) |
| "None", " " or "" |  | nothing |
| '$...$' | m37 | Render the string using mathtext. E.g "$f$" for marker showing the letter f. |
| verts |  | A list of (x, y) pairs used for Path vertices. The center of the marker is located at (0,0) and the size is normalized, such that the created path is encapsulated inside the unit cell. |
| path |  | A [Path](https://matplotlib.org/3.1.1/api/path_api.html#matplotlib.path.Path) instance. |
| (numsides, style, angle) |  | The marker can also be a tuple (numsides, style, angle), which will create a custom, regular symbol.  numsides:  the number of sides  style:  the style of the regular symbol:   * 0: a regular polygon * 1: a star-like symbol * 2: an asterisk * 3: a circle (numsides and angle is ignored); deprecated.   angle:  the angle of rotation of the symbol |

Areachart

import numpy as np  
import matplotlib.pyplot as plt  
  
# Create data  
x=range(1,6)  
y1=[1,4,6,8,9]  
y2=[2,2,7,10,12]  
y3=[2,8,5,10,6]  
  
# Basic stacked area chart.  
cols=["red","blue","yellow"]  
plt.stackplot(x,y1, y2, y3, colors=cols, labels=['A','B','C'],alpha=0.5) //alpha-> luminance  
plt.legend(loc='upper center', bbox\_to\_anchor=(1.1, 0.8), shadow=True, ncol=1)

plt.show()

Line types

import matplotlib.pyplot as plt

import random as random

students = ["Jane","Joe","Beck","Tom","Sam",

            "Eva","Samuel","Jack","Dana","Ester",

            "Carla","Steve","Fallon","Liam","Culhane",

            "Candance","Ana","Mari","Steffi","Adam"]

marks=[]

for i in range(0,len(students)):

     marks.append(random.randint(0, 101))

plt.xlabel("Students")

plt.ylabel("Marks")

plt.title("CLASS RECORDS")

plt.plot(students, marks, color = 'green',

         linestyle = 'solid', linewidth=’12’, marker = 'o',

         markerfacecolor = 'red', markersize = 12)

|  |  |  |
| --- | --- | --- |
| **Style** | **Or** | |
| 'solid' (default) | '-' |  |
| 'dotted' | ':' |  |
| 'dashed' | '--' |  |
| 'dashdot' | '-.' |  |
| 'None' | '' or ' ' |  |