Matplotlib Tutorial:-

Matplotlib Pyplot

Pyplot:-
Most of the Matplotlib utilities lies under the "pyplot" submodule, and are usually imported under the plt alias:

ploting:-

plot():-
The plot() function is used to draw points (markers) in a diagram.
By default,i,e,,without mentioned it, the plot() function draws a line from point to point.

Example:-
1)Draw a line in a diagram from position (0,0) to position (6,250):

```
#Three lines to make our compiler able to draw:
import sys
import matplotlib                                          ###These 3 lines are basic keywords
matplotlib.use('Agg')

import matplotlib.pyplot as plt
import numpy as np

xpoints = np.array([0, 6])
ypoints = np.array([0, 250])

plt.plot(xpoints, ypoints)
plt.show()

#Two  lines to make our compiler able to draw:
plt.savefig(sys.stdout.buffer)                             ###These 2 lines are also same
sys.stdout.flush()
```

Result:-https://www.w3schools.com/python/trypython.asp?filename=demo_matplotlib_pyplot
,,
,,,

Default X-Points:-

If we do not specify the points in the x-axis, they will get the default values 0, 1, 2, 3, (etc. depending on the length of the y-points.
So, if we take the same example as above, and leave out the x-points, the diagram will look like this:

Example
Plotting without x-points:

```
import matplotlib.pyplot as plt
import numpy as np

ypoints = np.array([3, 8, 1, 10, 5, 7])       ##here x is not given so by defaultly
consider x as  0,1,2,3,4,5 wrt  y value

plt.plot(ypoints)
plt.show()
```

o/p is https://www.w3schools.com/python/img_matplotlib_plotting4.png
,,
,,

Markers:-

You can use the keyword argument "marker" to emphasize each point with a specified marker:Marker is a point representation, it will be many type:-

Marker	Description
'o'	Circle
'*'	Star
'.'	Point
','	Pixel
'x'	X
'X'	X (filled)
'+'	Plus
'P'	Plus (filled)
's'	Square
'D'	Diamond
'd'	Diamond (thin)
'p'	Pentagon
'H'	Hexagon
'h'	Hexagon
'v'	Triangle Down
'^'	Triangle Up
'<'	Triangle Left
'>'	Triangle Right
'1'	Tri Down
'2'	Tri Up
'3'	Tri Left
'4'	Tri Right
'|'	Vline
'_'	Hline

*)Default Marker:-

Only by mensining shape of marker

```
import matplotlib.pyplot as plt
import numpy as np

xpoints = np.array([1, 8])
ypoints = np.array([3, 10])
```

```
plt.plot(xpoints, ypoints, 'o')
plt.show()
```

o/p is https://www.w3schools.com/python/img_matplotlib_plot_o.png

*)"marker"
Mark each point with a circle:

```
import matplotlib.pyplot as plt
import numpy as np

ypoints = np.array([3, 8, 1, 10])

plt.plot(ypoints, marker = 'o')
plt.show()
```

o/p is https://www.w3schools.com/python/img_matplotlib_marker_o.png

similarly,Example
Mark each point with a star:

```
...
plt.plot(ypoints, marker = '*')
...
```

https://www.w3schools.com/python/img_matplotlib_marker_star.png

,,,

Format Strings:-

You can also use the shortcut string notation parameter to specify the marker.
This parameter is also called fmt, and is written with this syntax:-

"marker|line|color"

Color Reference:-

Color Syntax	Description
'r'	Red
'g'	Green
'b'	Blue
'c'	Cyan
'm'	Magenta
'y'	Yellow
'k'	Black
'w'	White

Line Reference:-

Line

```
Syntax  Description
'-'     Solid line
':'     Dotted line
'--'    Dashed line
'-.'    Dashed/dotted line
```

Note: If you leave out the line value in the fmt parameter(i,e,,"marker|line|color"), no line will be plottet.

1)Example:-leave the line value
Mark each point with a circle and color red and no line formate:

```
import matplotlib.pyplot as plt
import numpy as np

ypoints = np.array([3, 8, 1, 10])

plt.plot(ypoints, 'or')                               ##o--->circle & r---->red &
no line will created
plt.show()
```

o/p is https://try.w3schools.com/try_python_img.php?id=295125344

2)Example:-line value is specified:-
Mark each point with a circle with red color dotted line:

```
import matplotlib.pyplot as plt
import numpy as np

ypoints = np.array([3, 8, 1, 10])

plt.plot(ypoints, '*:b')                              ##*-->star, :--->dotted
lines & b---->blue color
plt.show()
```

o/p is https://try.w3schools.com/try_python_img.php?id=295126843

,,,

Marker Size:-

You can use the keyword argument "markersize" or the shorter version, "ms" to set the size of the markers:

Example
Set the size of the markers to 20:

```
import matplotlib.pyplot as plt
import numpy as np
```

```
ypoints = np.array([3, 8, 1, 10])

plt.plot(ypoints, marker = 'o', ms = 20)                  ###ms is the marker size
20
plt.show()
 o/p is https://www.w3schools.com/python/img_matplotlib_marker_o_20.png
```

                                                                          Marker
Color:-

1)You can use the keyword argument "markeredgecolor" or the shorter "mec" to set the color of the edge of the markers:

Example
Set the EDGE color to red:

```
import matplotlib.pyplot as plt
import numpy as np

ypoints = np.array([3, 8, 1, 10])

plt.plot(ypoints, marker = 'o', ms = 20, mec = 'r')             ##ms---->marker_size
& mec--->marker_edge_color is red ,it was only red in background
plt.show()
```

2)You can use the keyword argument "markerfacecolor" or the shorter "mfc" to set the color inside the edge of the markers:

Example
Set the FACE color to red:

```
import matplotlib.pyplot as plt
import numpy as np

ypoints = np.array([3, 8, 1, 10])

plt.plot(ypoints, marker = 'o', ms = 20, mfc = 'r')
##mfc---->marker_face_color=red
plt.show()
```

3)Use both the mec and mfc arguments to color of the entire marker:

Example
Set the color of both the edge and the face to red:

```
import matplotlib.pyplot as plt
import numpy as np

ypoints = np.array([3, 8, 1, 10])
```

plt.plot(ypoints, marker = 'o', ms = 20, mec = 'r', mfc = 'b')   ##both mec is red(border) & mfc are used here
plt.show()
,,,

Hexadecimal Colors:-

Hexadecimal color values are also supported in all browsers.
A hexadecimal color is specified with: #RRGGBB.
RR (red), GG (green) and BB (blue) are hexadecimal integers between 00 and FF specifying the intensity of the color.
Hexadecimal number:-(16 number):-0123456789abcdef
For example, #0000FF is displayed as blue, because the blue component is set to its highest value (FF) and the others are set to 00.
,,

Supported colors:-

predefined colors exaple:- hotpink,,ect
plt.plot(ypoints, marker = 'o', ms = 20, mec = 'hotpink', mfc = 'hotpink')
,,

Linestyle:-

You can use the keyword argument "linestyle", or shorter "ls", to change the style of the plotted line:
dotted can be written as :.
dashed can be written as --.

Example
Use a dotted line:

```
import matplotlib.pyplot as plt
import numpy as np

ypoints = np.array([3, 8, 1, 10])

plt.plot(ypoints, linestyle = 'dotted')                ## it is similar to abouve examples :
plt.show()
```

o/p is https://www.w3schools.com/python/img_matplotlib_line_dotted.png
,,,

Line Color:-

You can use the keyword argument color or the shorter "c" to set the color of the line:

Example
Set the line color to red:

```
import matplotlib.pyplot as plt
import numpy as np

ypoints = np.array([3, 8, 1, 10])

plt.plot(ypoints, color = 'r')
plt.show()
```
o/p is https://www.w3schools.com/python/img_matplotlib_line_red.png

,,

Line Width:-

You can use the keyword argument linewidth or the shorter lw to change the width of the line.

The value is a floating number, in points:

Example
Plot with a 20.5pt wide line:

```
import matplotlib.pyplot as plt
import numpy as np

ypoints = np.array([3, 8, 1, 10])

plt.plot(ypoints, linewidth = '20.5')
plt.show()
```
o/p is https://www.w3schools.com/python/img_matplotlib_line_lw.png

,,

Multiple Lines:-

You can plot as many lines as you like by simply adding more plt.plot() functions:

Example
Draw two lines by specifying a "plt.plot()" function for each line:

```
import matplotlib.pyplot as plt
import numpy as np

y1 = np.array([3, 8, 1, 10])
y2 = np.array([6, 2, 7, 11])
```

```
plt.plot(y1)                                            ##it will produce 1st graph
y1 wrt default value of x
plt.plot(y2)                                            ##it will produce 2nd graph
y2 wrt default value of x

plt.show()
o/p is https://www.w3schools.com/python/img_matplotlib_line_two.png
```

note:-
You can also plot many lines by adding the points for the x- and y-axis for each line in the same plt.plot() function.
(In the examples above we only specified the points on the y-axis, meaning that the points on the x-axis got the the default values (0, 1, 2, 3).)

,,

Matplotlib Labels and Title:-

1)Create Labels for a Plot:-

With Pyplot, you can use the "xlabel()" and "ylabel()" functions to set a label for the x- and y-axis.

Example
1)Add labels to the x- and y-axis:

```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

plt.plot(x, y)

plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")

plt.show()
o/p is https://www.w3schools.com/python/img_matplotlib_labels.png
```

2)Create a Title for a Plot:-

With Pyplot, you can use the "title()" function to set a title for the plot.

Example
Add a plot title and labels for the x- and y-axis:

```
import numpy as np
import matplotlib.pyplot as plt
```

```
x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

plt.plot(x, y)

plt.title("Sports Watch Data")
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")

plt.show()
```

o/p is https://www.w3schools.com/python/img_matplotlib_title.png

,,,

Set Font Properties for Title and Labels:-

1)You can use the "fontdict" parameter in xlabel(), ylabel(), and title() to set font properties for the title and labels.fontdict is split into font+dictionary

Example
Set font properties for the title and labels:

```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

font1 = {'family':'serif','color':'blue','size':20}
##Distnary={keys:values}
font2 = {'family':'serif','color':'darkred','size':15}
##keys are 'family', 'color' ,'size'

plt.title("Sports Watch Data", fontdict = font1)
plt.xlabel("Average Pulse", fontdict = font2)
plt.ylabel("Calorie Burnage", fontdict = font2)

plt.plot(x, y)
plt.show()
```

o/p is https://www.w3schools.com/python/img_matplotlib_title_fontdict.png

2)Position the Title:-
You can use the "loc" parameter in title() to position the title.

Legal values are: 'left', 'right', and 'center'. Default value is 'center'.

Example
Position the title to the left:

```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

plt.title("Sports Watch Data", loc = 'left')                            ###Title
should be in the left side
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")

plt.plot(x, y)
plt.show()
```

o/p is https://www.w3schools.com/python/img_matplotlib_title_loc.png
----

GRID LINES:-

1)Add Grid Lines to a Plot:-

With Pyplot, you can use the "grid()" function to add grid lines to the plot.

Example:-
Add grid lines to the plot:

```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

plt.title("Sports Watch Data")
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")

plt.plot(x, y)
plt.grid()                                              ##Grid are added to the
graphs
plt.show()
```

o/p is https://www.w3schools.com/python/img_matplotlib_grid.png

2)Specify Which Grid Lines to Display(it specifies grids are appiled on which axis either x or y):-

You can use the axis parameter in the "grid()" function to specify which grid lines to display.

Legal values are: 'x', 'y', and 'both'. Default value is 'both'.

Example
Display only grid lines for the x-axis:

```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

plt.title("Sports Watch Data")
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")

plt.plot(x, y)
plt.grid(axis = 'x')                            ##it willl give only grids on x axis
plt.show()
```

        3)Set Line Properties for the Grid:-

You can also set the line properties of the grid, like this: grid(color = 'color', linestyle = 'linestyle', linewidth = number).

Example
Set the line properties of the grid:

```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

plt.title("Sports Watch Data")
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")

plt.plot(x, y)
plt.grid(color = 'green', linestyle = '--', linewidth = 0.5)
plt.show()
```


SUBPLOTS:-

        The subplots() Function:-
The "subplots()" function takes three arguments that describes the layout of the figure.
The layout is organized in rows and columns, which are represented by the first and second argument.

The third argument represents the index of the current plot.

        1)Display Multiple Plots:-

With the "subplots()" function you can draw multiple plots in one figure:

Example
1)Draw 2 plots:

```
import matplotlib.pyplot as plt
import numpy as np

#plot 1:
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(1, 2, 1)
plt.title("SALES")                                               ##subplot(row,
column ,plot_number)---->subplot(1,2,1),means 1 row ,2 columnn's & it is the 1st
figure
plt.plot(x,y)                                                      #the figure has 1
row, 2 columns, and this plot is the first plot.

#plot 2:
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(1, 2, 2)
plt.title("INCOME")                                      ##subplot(row, column
,plot_number)---->subplot(1,2,1),means 1 row ,2 columnn's & it is the 2nd figure
plt.plot(x,y)                                             #the figure has 1 row, 2
columns, and this plot is the second plot.

plt.show()
```

o/p is https://www.w3schools.com/python/img_matplotlib_subplots1.png   ,So, if we want a figure with 2 rows an 1 column (meaning that the two plots will be displayed on top of each other instead of side-by-side), we can write the syntax like this:

        2)Multi_plots:-

you can draw as many plots you like on one figure, just descibe the number of rows, columns, and the index of the plot.

Example
Draw 6 plots:

```
import matplotlib.pyplot as plt
import numpy as np
```

```
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(2, 3, 1)                                    ##2 rows ,3 columns  & 1st
figure
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(2, 3, 2)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(2, 3, 3)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(2, 3, 4)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(2, 3, 5)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(2, 3, 6)
plt.plot(x,y)

plt.show()
```
o/p is https://www.w3schools.com/python/img_matplotlib_subplots3.png

3)Super Title:-

In the case of "title" we get only title for one perticular plot, but in the case of multi_plot we need to use Super title for entire sheet of containing all the graphs.
You can add a title to the entire figure with the "suptitle()" function:

Example
Add a title for the entire figure:

```
import matplotlib.pyplot as plt
import numpy as np

#plot 1:
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(1, 2, 1)
plt.plot(x,y)
plt.title("SALES")

#plot 2:
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(1, 2, 2)
plt.plot(x,y)
plt.title("INCOME")

plt.suptitle("MY SHOP")                                  ##suptitle---->super title
plt.show()
o/p is https://www.w3schools.com/python/img_matplotlib_subplots5.png
```

SCATTER PLOT:-

Creating Scatter Plots:-

With Pyplot, you can use the "scatter()" function to draw a scatter plot.
The scatter() function plots one dot for each observation. It needs two arrays of the same length, one for the values of the x-axis, and one for values on the y-axis:

Example
A simple scatter plot:

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])

plt.scatter(x, y)
plt.show()
O/P IS https://www.w3schools.com/python/img_matplotlib_scatter.png
```
OBSERVATIONS:-The observation in the example above is the result of 13 cars passing by.
The X-axis shows how old the car is.
The Y-axis shows the speed of the car when it passes.
Are there any relationships between the observations?

It seems that the newer the car, the faster it drives, but that could be a coincidence, after all we only registered 13 cars.

2)Compare plot

In the example above, there seems to be a relationship between speed and age, but what if we plot the observations from another day as well(for getting conformation)? Will the scatter plot tell us something else?
Therefore we need to compare the things with other

Example
Draw two scatter plots on the same figure for comparing:

```
import matplotlib.pyplot as plt
import numpy as np

#day one, the age and speed of 13 cars:
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
plt.scatter(x, y)                                                       ##1st
scatter plot

#day two, the age and speed of 15 cars:
x = np.array([2,2,8,1,15,8,12,9,7,3,11,4,7,14,12])
y = np.array([100,105,84,105,90,99,90,95,94,100,79,112,91,80,85])
plt.scatter(x, y)                                                       ##2nd
scatter plot
plt.show()

##If we won't specify the different color for 2 plots by defaultly it  will create 2
different color for different  experiments
```

o/p is https://www.w3schools.com/python/img_matplotlib_scatter_compare.png, By comparing the two plots, I think it is safe to say that they both gives us the same conclusion: the newer the car, the faster it drives.

3)Colors:-

You can set your own color for each scatter plot with the color or the "c" argument:

Example
Set your own color of the markers:

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
plt.scatter(x, y, color = 'hotpink')

x = np.array([2,2,8,1,15,8,12,9,7,3,11,4,7,14,12])
```

```
y = np.array([100,105,84,105,90,99,90,95,94,100,79,112,91,80,85])
plt.scatter(x, y, color = '#88c999')

plt.show()
o/p https://www.w3schools.com/python/img_matplotlib_scatter_color.png
```

        4)Color Each Dot:-

You can even set a specific color for each dot by using an array of colors as value for the c argument:
Note: You cannot use the color argument for this, only the c argument.(bcz in the syntax we have to use color argument as c as seen in the above example)

Example:-
Set your own color of the markers:

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
colors =
np.array(["red","green","blue","yellow","pink","black","orange","purple","beige","br
own","gray","cyan","magenta"]) #it will assign colors for each individuals

plt.scatter(x, y, c=colors)

plt.show()
```

        5)ColorMap:-

The Matplotlib module has a number of available colormaps.
A colormap is like a list of colors, where each color has a value that ranges from 0 to 100.
Here is an example of a colormap:This colormap is called 'viridis' and as you can see it ranges from 0, which is a purple color, and up to 100, which is a yellow color.
Example:-

```
mport matplotlib.pyplot as plt
import numpy as np

x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
colors = np.array([0, 10, 20, 30, 40, 45, 50, 55, 60, 70, 80, 90, 100])

plt.scatter(x, y, c=colors, cmap='viridis')

plt.show()
```

        6)Size of SCATTER POIINT:-

You can change the size of the dots with the "s" argument.

Just like colors, make sure the array for sizes has the same length as the arrays for the x- and y-axis:

Example
Set your own size for the markers:

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
sizes = np.array([20,50,100,200,500,1000,60,90,10,300,600,800,75])

plt.scatter(x, y, s=sizes)                                          ##IN marker
polt we assign size as "ms"

plt.show()
```

7)Alpha:-

You can adjust the "transparency of the dots" with the alpha argument.
Just like colors, make sure the array for sizes has the same length as the arrays for the x- and y-axis:

Example
Set your own size for the markers:

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
sizes = np.array([20,50,100,200,500,1000,60,90,10,300,600,800,75])

plt.scatter(x, y, s=sizes, alpha=0.5)
plt.show()
```

8)Combine Color Size and Alpha:-

You can combine a colormap with different sizes on the dots. This is best visualized if the dots are transparent:

Example
Create random arrays with 100 values for x-points, y-points, colors and sizes:

```
import matplotlib.pyplot as plt
import numpy as np
```

```
x = np.random.randint(100, size=(100))
y = np.random.randint(100, size=(100))
colors = np.random.randint(100, size=(100))
sizes = 10 * np.random.randint(100, size=(100))                 ##random value is
multiplied with 10 it gives  size of each ramdom integers of 100.

plt.scatter(x, y, c=colors, s=sizes, alpha=0.5, cmap='nipy_spectral')

plt.colorbar()

plt.show()
```

o/p is https://www.w3schools.com/python/img_matplotlib_scatter_combine.png

----

BAR GRAPH:-

1)Creating Bars:-

With Pyplot, you can use the "bar()" function to draw bar graphs:
The bar() function takes arguments that describes the layout of the bars.

Example
Draw 4 bars:

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.bar(x,y)
plt.show()
```

O/P IS https://www.w3schools.com/python/img_matplotlib_bars1.png

2)Horizontal Bars:-

If you want the bars to be displayed horizontally instead of vertically, use the "barh()" function:barh means bars in horizontal

Example
Draw 4 horizontal bars:

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array(["A", "B", "C", "D"])
```

```
y = np.array([3, 8, 1, 10])

plt.barh(x, y)
plt.show()
```
o/p is https://www.w3schools.com/python/img_matplotlib_bars2.png

3)BAR COLOR:-

The "bar() and barh()" takes the keyword argument color to set the color of the bars:

Example
Draw 4 red bars:

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.bar(x, y, color = "red")                            ##COLOR = RED
plt.show()
```

4)BAR WIDTH:-The default(without specifing) bar width is "0.8"

The bar() takes the keyword argument width to set the width of the bars.

Example
Draw 4 very thin bars:

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.bar(x, y, width = 0.1)                              ##0.1 gives very thin width
size.
plt.show()
```

Note: For horizontal bars, use height instead of width.
i,e,,plt.barh(x, y, height = 0.1)
The default height value is "0.8"

HISTOGRAM:-

Histogram:-
A histogram is a graph showing frequency distributions.

It is a graph showing the number of observations within each given interval.

1)Create Histogram:-
In Matplotlib, we use the "hist()" function to create histograms.
The "hist()" function will use an array of numbers to create a histogram, the array is sent into the function as an argument.

Example:
Say you ask for the height of 250 people, you might end up with a histogram like this:
For simplicity we use NumPy to randomly generate an array with 250 values, where the values will concentrate around 170 means mean=170, and the standard deviation is 10.

The hist() function will read the array and produce a histogram:

Example:-
A simple histogram:

```
#Three lines to make our compiler able to draw:
import sys
import matplotlib
matplotlib.use('Agg')

import matplotlib.pyplot as plt
import numpy as np

x = np.random.normal(170, 10, 250)                      ##it will produse random
number  of 250 numbers at 170 mean position and standard_deviation of 10

plt.hist(x)                                             ##hist() will produce
histogram graph about these values

plt.show()

#Two  lines to make our compiler able to draw:
plt.savefig(sys.stdout.buffer)
sys.stdout.flush()
```

o/p is https://www.w3schools.com/python/img_matplotlib_histogram1.png

----

PIE CHART:-

1)Creating Pie Charts:-
With Pyplot, you can use the "pie()" function to draw pie charts:

Example
A simple pie chart:

```
import matplotlib.pyplot as plt
import numpy as np

y = np.array([35, 25, 25, 15])

plt.pie(y)
plt.show()
```

O/P IS https://www.w3schools.com/python/img_matplotlib_pie1.png

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(i,e,,at 0 degree) and move counterclockwise:

Note: The size of each wedge is determined by comparing the value with all the other values, by using this formula:
The value divided by the sum of all values: x/sum(x)

2)Labels:-

Add labels to the pie chart with the label parameter.
The label parameter must be an array with one label for each wedge:

Example
A simple pie chart:

```
import matplotlib.pyplot as plt
import numpy as np

y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]

plt.pie(y, labels = mylabels)
plt.show()
```
o/p is https://www.w3schools.com/python/img_matplotlib_pie_labels.png

3)Start Angle:-

As mentioned the default start angle is at the x-axis, but you can change the start angle by specifying a startangle parameter.
The startangle parameter is defined with an angle in degrees, default angle is 0:

Example:-

Start the first wedge at 90 degrees:

```
import matplotlib.pyplot as plt
import numpy as np
```

```
y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]

plt.pie(y, labels = mylabels, startangle = 90)
plt.show()
o/p is https://www.w3schools.com/python/img_matplotlib_pie_angle_90.png
```

        4)Explode:-

Maybe you want one of the wedges to stand out? The explode parameter allows you to do that.
The explode parameter, if specified, and not None, must be an array with one value for each wedge.

Each value represents how far from the center each wedge is displayed:

Example
Pull the "Apples" wedge 0.2 from the center of the pie:

```
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]                                             ##Here
Apples wedge is exposed from 0.2 distance from all pie wedge

plt.pie(y, labels = mylabels, explode = myexplode)
plt.show()
o/p is https://www.w3schools.com/python/img_matplotlib_pie_explode.png
```

        5)Shadow:-

Add a "shadow" to the pie chart by setting the shadows parameter to True:

Example
Add a shadow:

```
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, explode = myexplode, shadow = True)
##If it shows true, it shows shadow & if it shows false it won't shows shadow
plt.show()
```

        6)Colors:-

You can set the color of each wedge with the colors parameter.
The colors parameter, if specified, must be an array with one value for each wedge:

Example:-
Specify a new color for each wedge:

```
import matplotlib.pyplot as plt
import numpy as np

y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]
mycolors = ["black", "hotpink", "b", "#4CAF50"]                         ##it shows
color for each wedge

plt.pie(y, labels = mylabels, colors = mycolors)
plt.show()
```

o/p is https://www.w3schools.com/python/img_matplotlib_pie_color.png

        7)a)Legend:-

To add a list of explanation for each wedge, use the "legend()" function:It is used for user identification.

Example
Add a legend:

```
import matplotlib.pyplot as plt
import numpy as np

y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]

plt.pie(y, labels = mylabels)
plt.legend()                                          ##it will shows "Apples",
"Bananas", "Cherries", "Dates" in the chart wrt to color for user identification
plt.show()
```

        b)Legend With Header:-

To add a header to the legend, add the title parameter to the legend function.it give title to legend

Example
Add a legend with a header:

```
import matplotlib.pyplot as plt
import numpy as np

y = np.array([35, 25, 25, 15])
```

```
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]

plt.pie(y, labels = mylabels)
plt.legend(title = "Four Fruits:")
plt.show()
```

o/p is https://www.w3schools.com/python/img_matplotlib_pie_legend_title.png