Matplotlib Tutorial

What is Matplotlib?

Matplotlib is a low level graph plotting library in python that serves as a visualization utility.
Matplotlib was created by John D. Hunter.
Matplotlib is open source and we can use it freely.
Matplotlib is mostly written in python, a few segments are written in C, Objective-C and Javascript for Platform compatibility.

Matplotlib Getting Started

Installation of Matplotlib

If you have Python and PIP already installed on a system, then installation of Matplotlib is very easy.
Install it using this command:
C:\Users*Your Name*>pip install matplotlib
If this command fails, then use a python distribution that already has Matplotlib installed, like Anaconda, Spyder etc.

Import Matplotlib

Once Matplotlib is installed, import it in your applications by adding
the import *module* statement:
import matplotlib
Now Matplotlib is imported and ready to use:

Checking Matplotlib Version

The version string is stored under __version__ attribute.

Example

```
import matplotlib

print(matplotlib.__version__)
```

Note: two underscore characters are used in __version__.

Matplotlib Pyplot

Pyplot

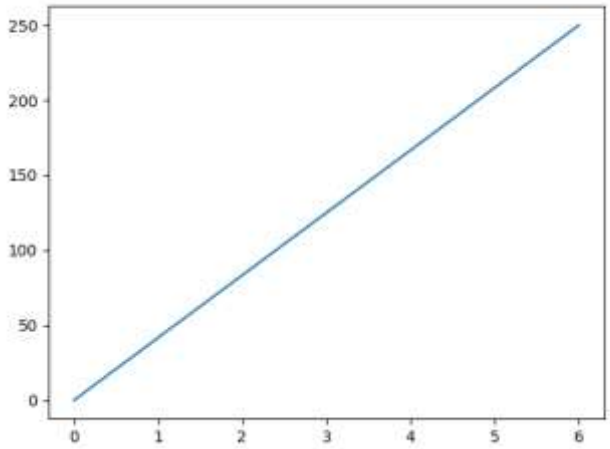
Most of the Matplotlib utilities lies under the pyplot submodule, and are usually imported under the plt alias:
import matplotlib.pyplot as plt
Now the Pyplot package can be referred to as plt.

Example

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

```
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()
```

Result:



Matplotlib Plotting

Plotting x and y points

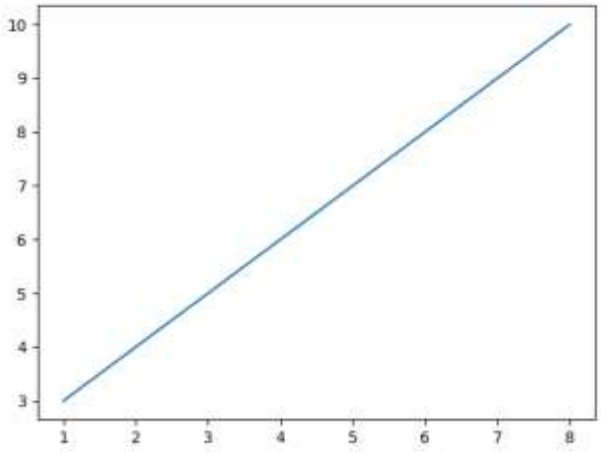
The plot() function is used to draw points (markers) in a diagram.
By default, the plot() function draws a line from point to point.
The function takes parameters for specifying points in the diagram.
Parameter 1 is an array containing the points on the **x-axis**.
Parameter 2 is an array containing the points on the **y-axis**.
If we need to plot a line from (1, 3) to (8, 10), we have to pass two arrays [1, 8] and [3, 10] to the plot function.

Example

Draw a line in a diagram from position (1, 3) to position (8, 10):

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

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

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

Result:



The **x-axis** is the horizontal axis.
The **y-axis** is the vertical axis.

Plotting Without Line

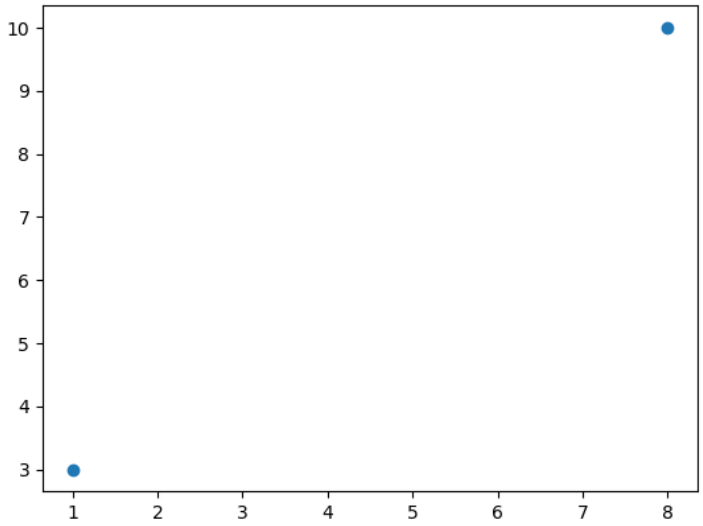
To plot only the markers, you can use *shortcut string notation* parameter 'o', which means 'rings'.

Example

Draw two points in the diagram, one at position (1, 3) and one in position (8, 10):

```
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()
```

Result:



You will learn more about markers in the next chapter.

Multiple Points

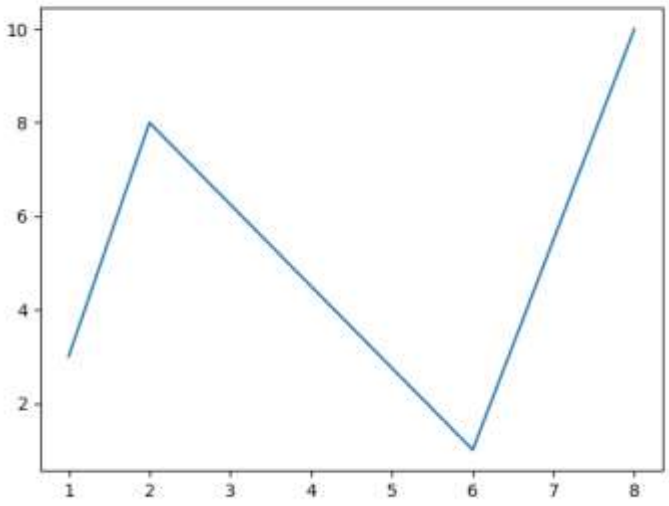
You can plot as many points as you like, just make sure you have the same number of points in both axis.

Example

Draw a line in a diagram from position (1, 3) to (2, 8) then to (6, 1) and finally to position (8, 10):

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

xpoints = np.array([1, 2, 6, 8])
ypoints = np.array([3, 8, 1, 10])

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

Result:



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.

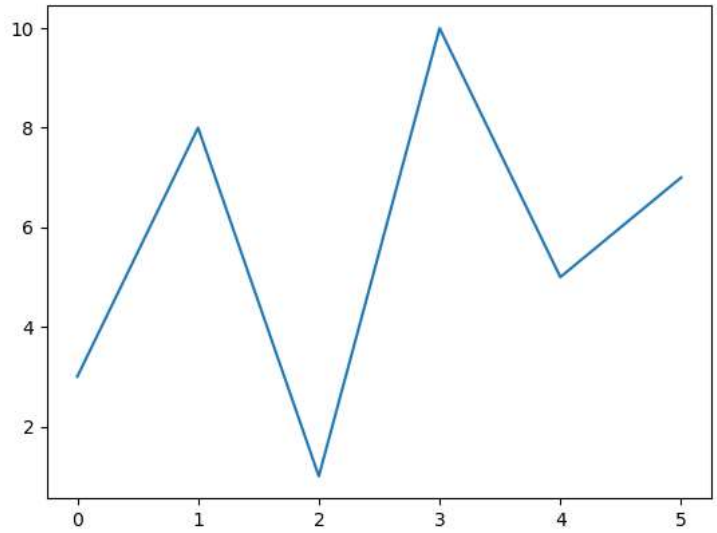
So, if we take the same example as above, and leave out the x-points, the diagram will look like this:

Example

Plotting without x-points:

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

ypoints = np.array([3, 8, 1, 10, 5, 7])

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

Result:



The **x-points** in the example above is [0, 1, 2, 3, 4, 5].

Matplotlib Markers

Markers

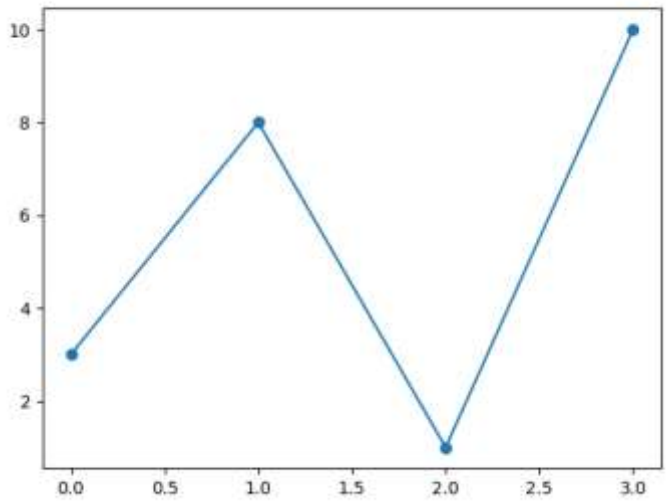
You can use the keyword argument marker to emphasize each point with a specified marker:

Example

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()
```

Result:

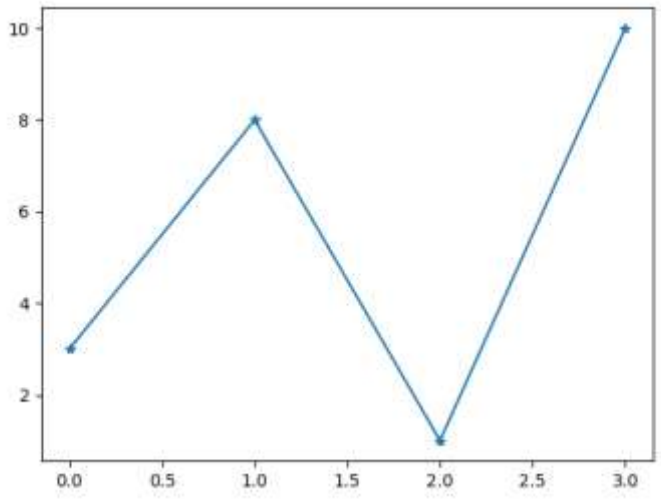


Example

Mark each point with a star:

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

Result:



Marker Reference

You can choose any of these markers:

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

Format Strings fmt

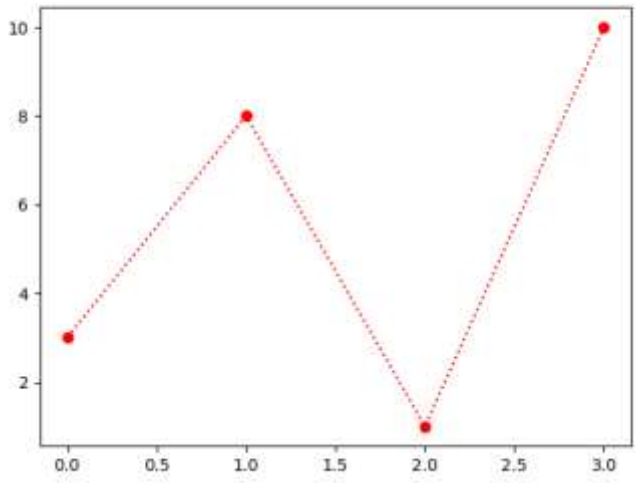
You can use also use the *shortcut string notation* parameter to specify the marker.
This parameter is also called fmt, and is written with this syntax:
marker|line|color

Example

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, 'o:r')
plt.show()
```

Result:



The marker value can be anything from the Marker Reference above.
The line value can be one of the following:

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, no line will be plottet.
The short color value can be one of the following:

Color Reference

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

Marker Size

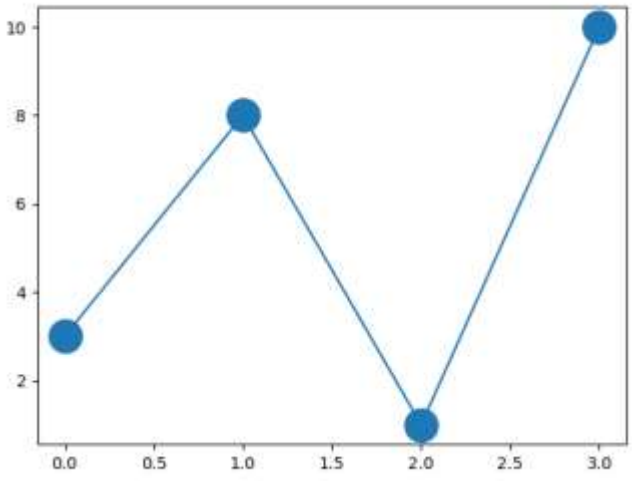
You can use the keyword argument markersize or the shorter version, ms to set the size of the markers:

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)
plt.show()
```

Result:



Marker Color

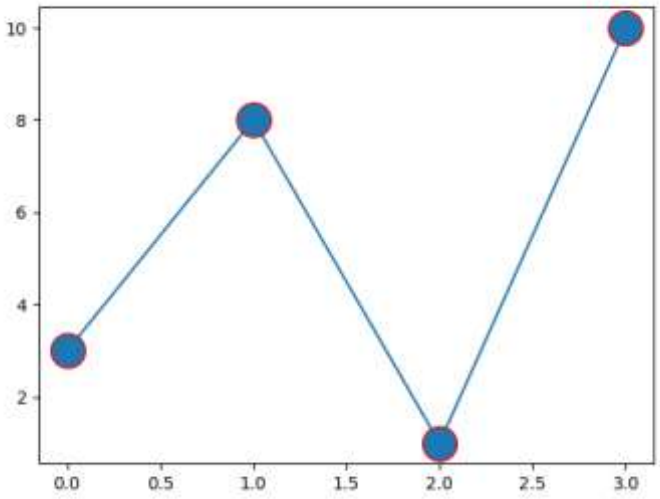
You can use the keyword argument markeredgecolor or the shorter mec to set the color of the *edge* of the markers:

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')
plt.show()
```

Result:



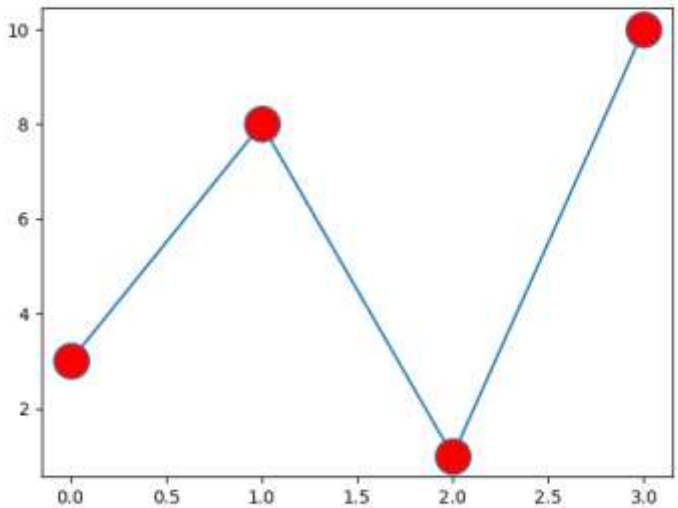
You can use the keyword argument markerfacecolor or the shorter mfc to set the color inside the edge of the markers:

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')
plt.show()
```

Result:



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

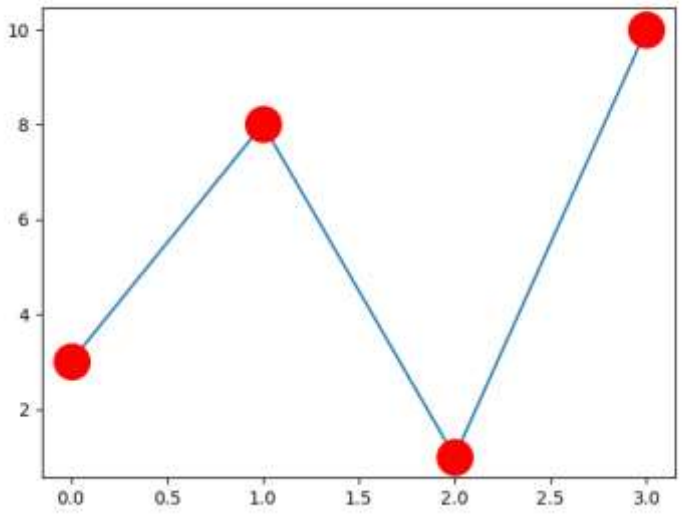
Example

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 = 'r')
plt.show()
```

Result:



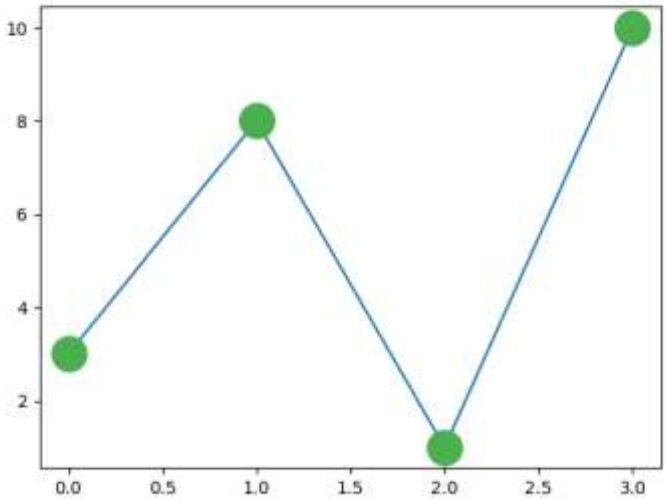
You can also use Hexadecimal color values:

Example

Mark each point with a beautiful green color:

```
...
plt.plot(ypoints, marker = 'o', ms = 20, mec = '#4CAF50', mfc = '#4CAF50')
...
```

Result:



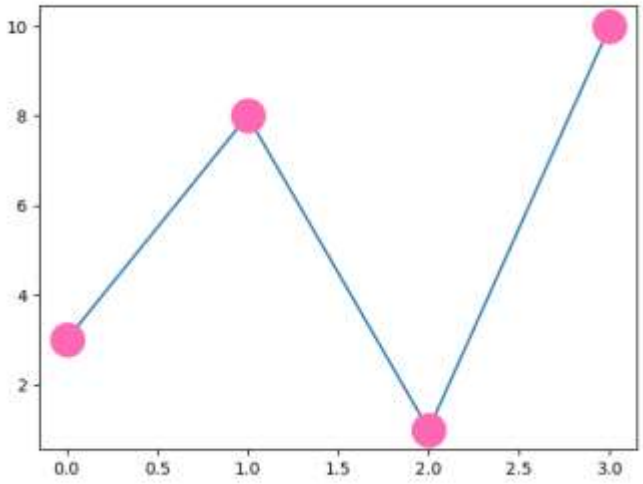
Or any of the 140 supported color names.

Example

Mark each point with the color named "hotpink":

```
...
plt.plot(ypoints, marker = 'o', ms = 20, mec = 'hotpink', mfc = 'hotpink')
...
```

Result:



Matplotlib Line

Linestyle

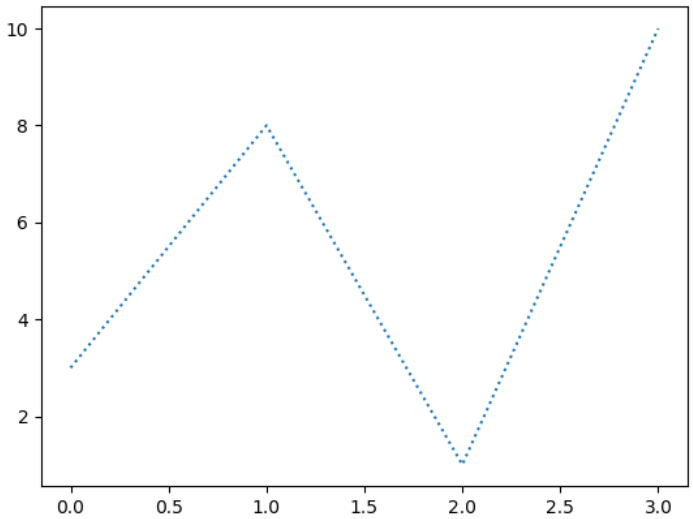
You can use the keyword argument linestyle, or shorter ls, to change the style of the plotted line:

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')
plt.show()
```

Result:

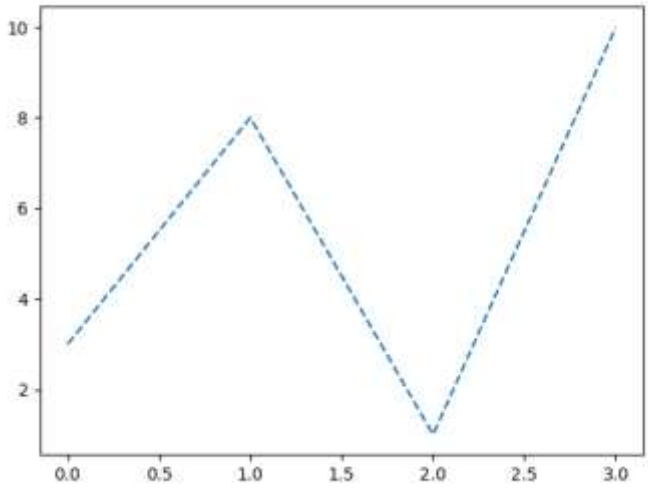


Example

Use a dashed line:

plt.plot(ypoints, linestyle = 'dashed')

Result:



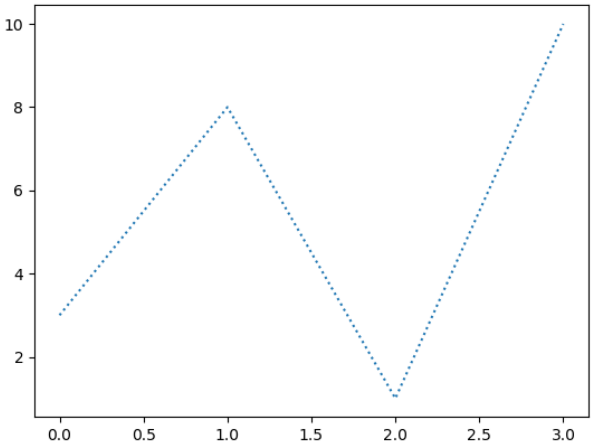
Shorter Syntax

The line style can be written in a shorter syntax:
linestyle can be written as ls.
dotted can be written as :.
dashed can be written as --.

Example

Shorter syntax:
plt.plot(ypoints, ls = ':')

Result:



Line Styles

You can choose any of these styles:

Style	Or
'solid' (default)	'-'

'dotted'	':'
'dashed'	'--'
'dashdot'	'-.'
'None'	'' or ' '

Line Color

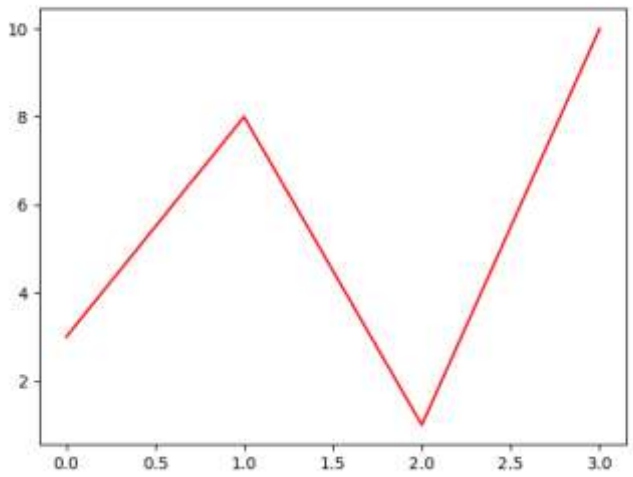
You can use the keyword argument color or the shorter c to set the color of the line:

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()
```

Result:



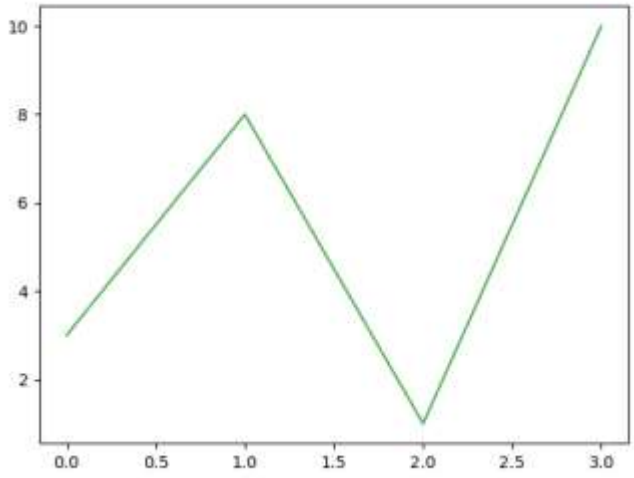
You can also use Hexadecimal color values:

Example

Plot with a beautiful green line:

```
...
plt.plot(ypoints, c = '#4CAF50')
...
```

Result:



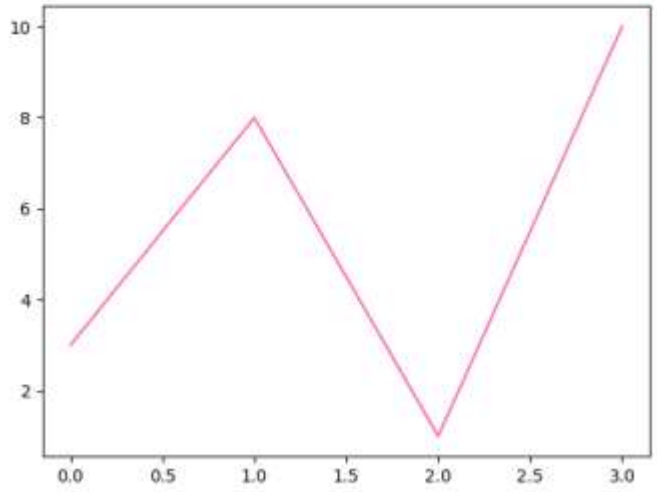
Or any of the 140 supported color names.

Example

Plot with the color named "hotpink":

```
...
plt.plot(ypoints, c = 'hotpink')
...
```

Result:



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:

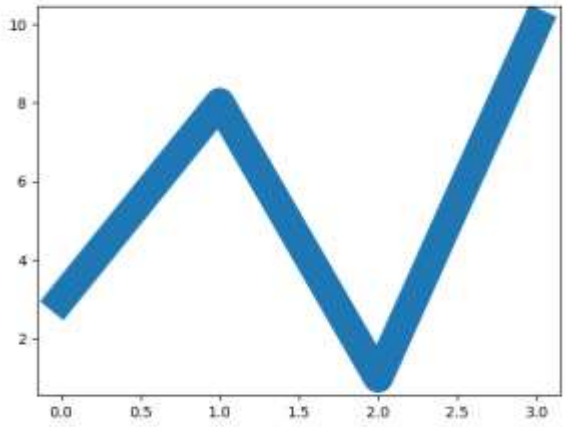
Example

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()
```

Result:



Multiple Lines

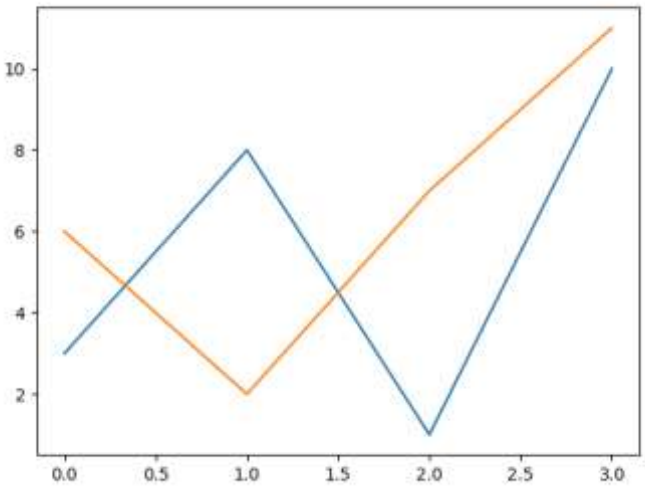
You can plot as many lines as you like by simply adding more plt.plot() functions:

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)
plt.plot(y2)

plt.show()
```

Result:



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.

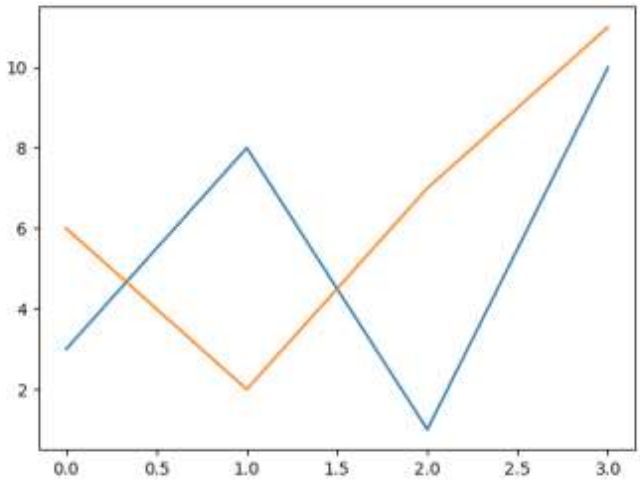
(In the examples above we only specified the points on the y-axis, meaning that the points on the x-axis got the the default values (0, 1, 2, 3).)
The x- and y- values come in pairs:

Example

Draw two lines by specifiyng the x- and y-point values for both lines:

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

x1 = np.array([0, 1, 2, 3])
y1 = np.array([3, 8, 1, 10])
x2 = np.array([0, 1, 2, 3])
y2 = np.array([6, 2, 7, 11])

plt.plot(x1, y1, x2, y2)
plt.show()
```

Result:



Matplotlib Labels and Title

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.

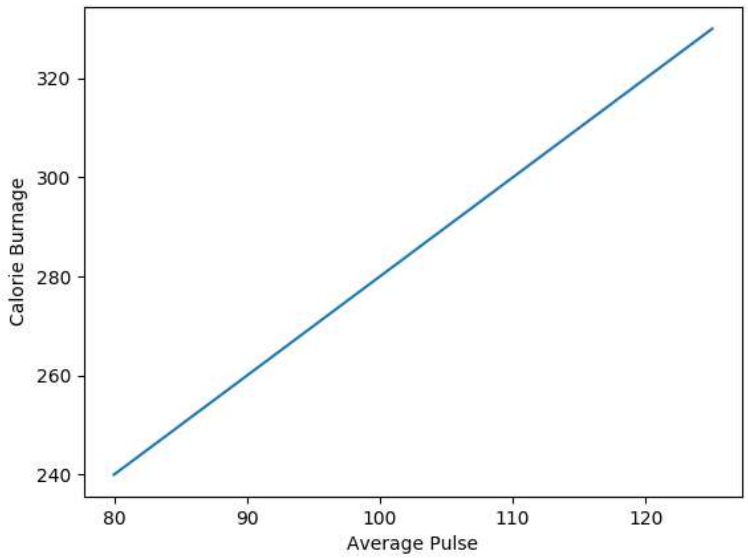
Example

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()

Result:



Create a Title for a Plot

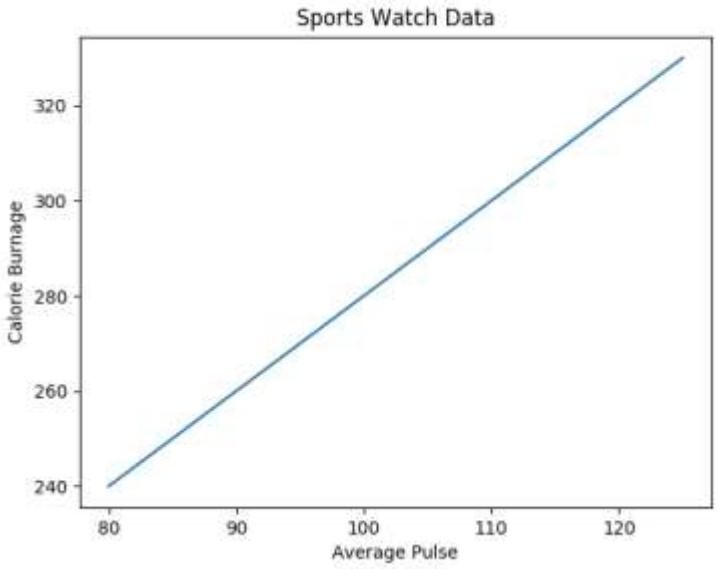
With Pyplot, you can use the title() function to set a title for the plot.

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()
```

Result:



Set Font Properties for Title and Labels

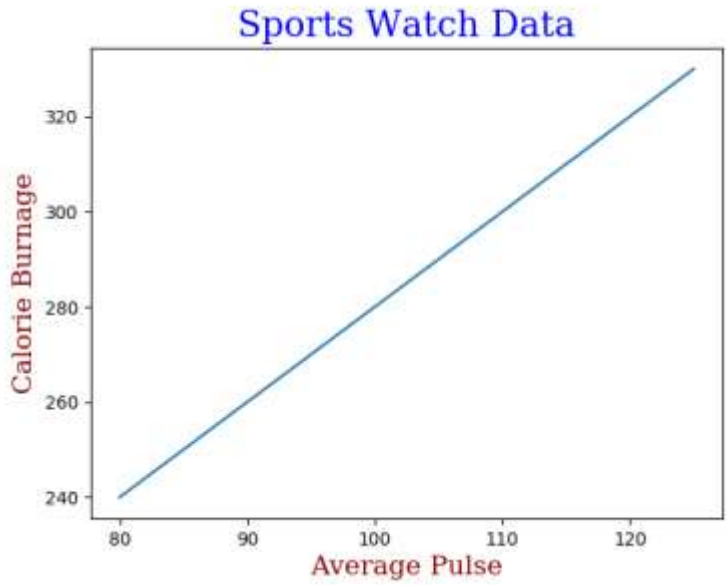
You can use the fontdict parameter in xlabel(), ylabel(), and title() to set font properties for the title and labels.

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}
font2 = {'family':'serif','color':'darkred','size':15}

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()
```

Result:



Position the Title

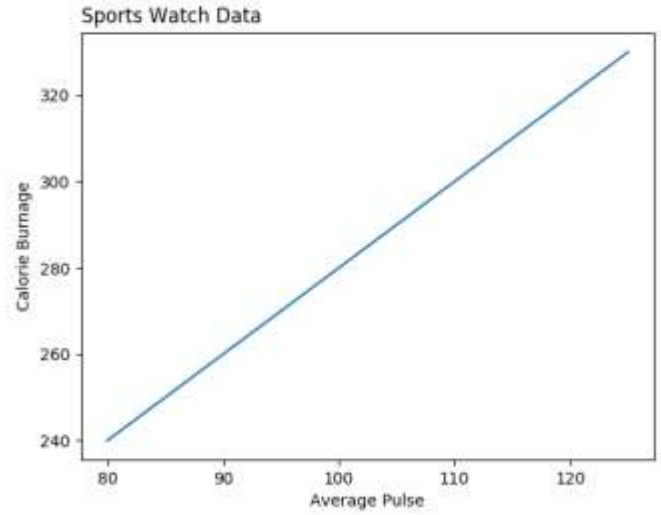
You can use the loc parameter in title() to position the title.
Legal values are: 'left', 'right', and 'center'. Default value is 'center'.

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

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

Result:



Matplotlib Adding Grid Lines

Add Grid Lines to a Plot

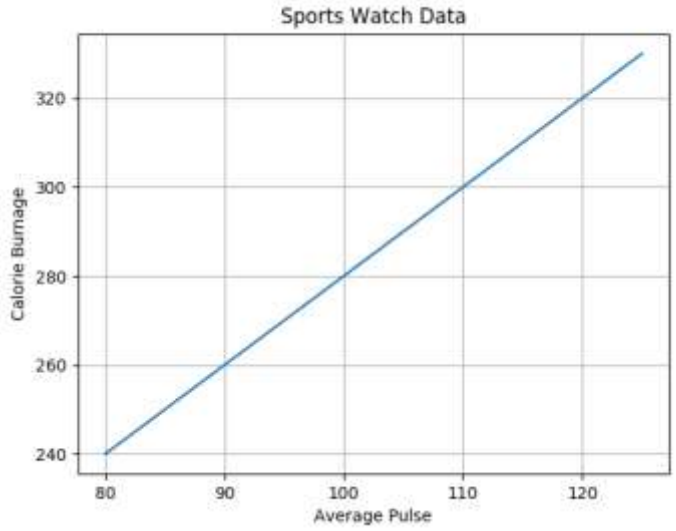
With Pyplot, you can use the grid() function to add grid lines to the plot.

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()

plt.show()
```

Result:



Specify Which Grid Lines to Display

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'.

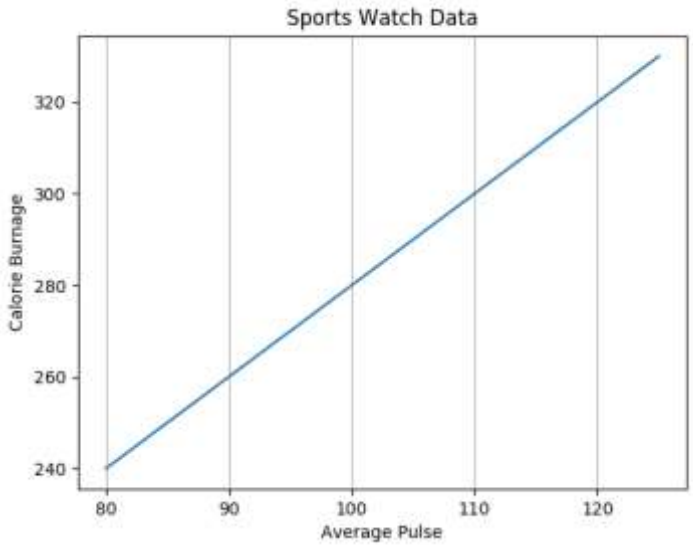
Example

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')

plt.show()
```

Result:

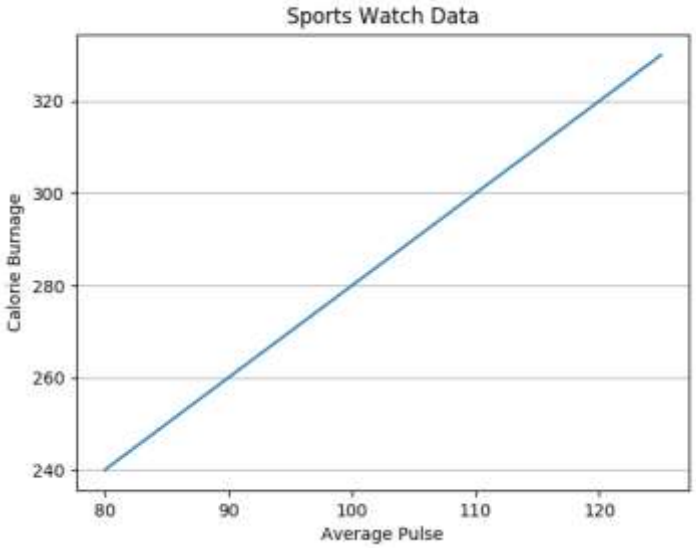


Example

Display only grid lines for the 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.title("Sports Watch Data")
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")

plt.plot(x, y)

plt.grid(axis = 'y')

plt.show()
```

Result:



Set Line Properties for the Grid

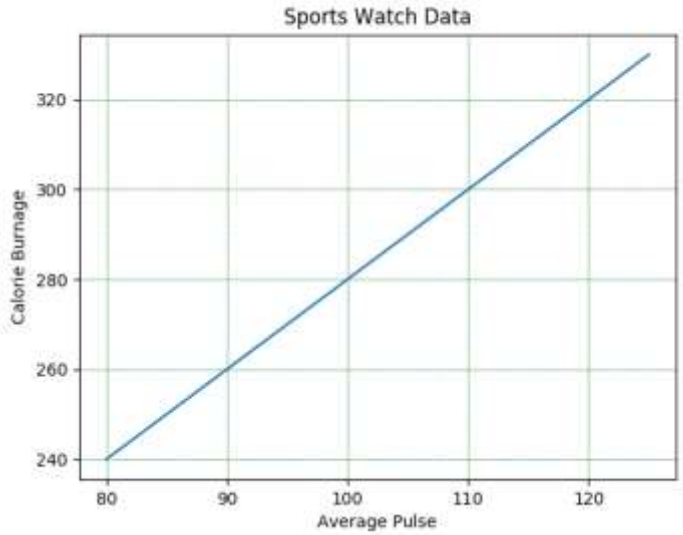
You can also set the line properties of the grid, like this: grid(color = '*color*', linestyle = '*linestyle*', linewidth = *number*).

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()
```

Result:



Matplotlib Subplots

Display Multiple Plots

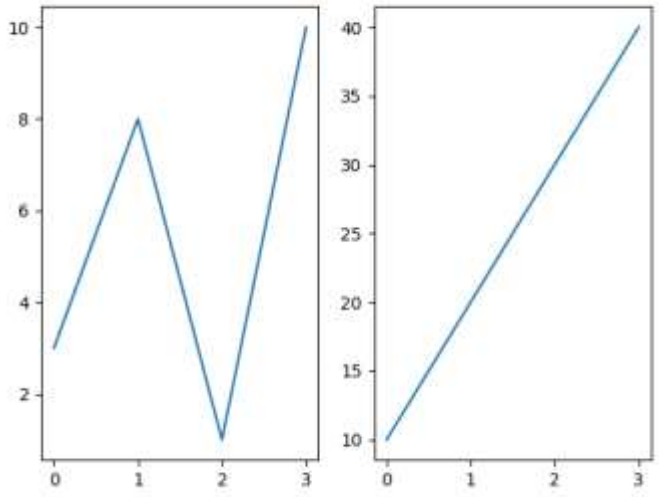
With the subplots() function you can draw multiple plots in one figure:

Example

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.plot(x,y)

#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.show()
```

Result:



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.

```
plt.subplot(1, 2, 1)
#the figure has 1 row, 2 columns, and this plot is the first plot.

plt.subplot(1, 2, 2)
#the figure has 1 row, 2 columns, and this plot is the second plot.
```

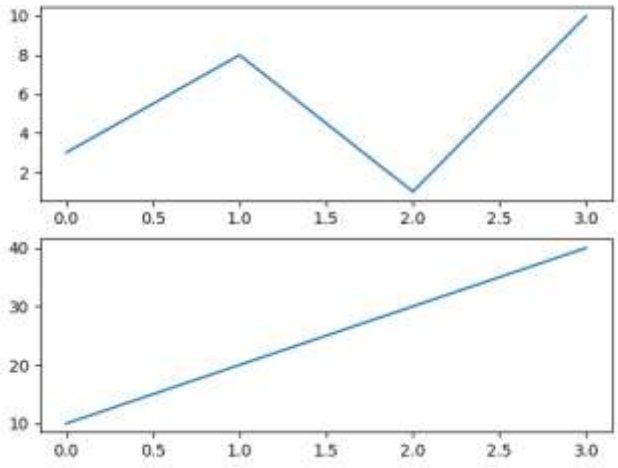
So, if we want a figure with 2 rows an 1 column (meaning that the two plots will be displayed on top of each other instead of side-by-side), we can write the syntax like this:

Example

Draw 2 plots on top of each other:

```
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(2, 1, 1)
plt.plot(x,y)

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

plt.subplot(2, 1, 2)
plt.plot(x,y)

plt.show()
```

Result:



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

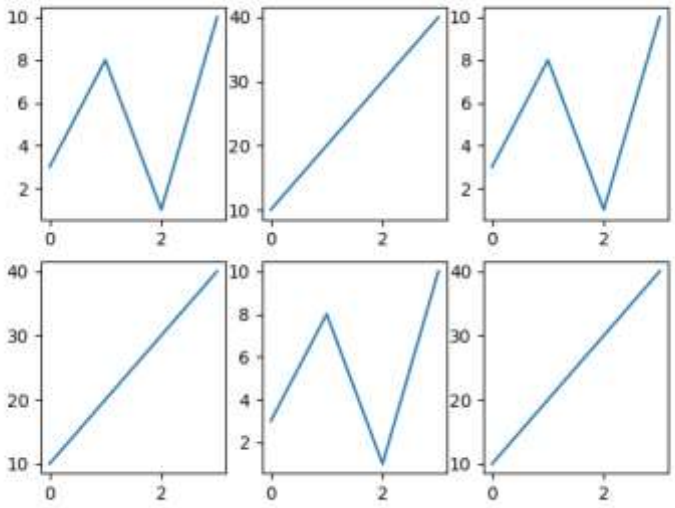
Example

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)
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()
```

Result:



Title

You can add a title to each plot with the title() function:

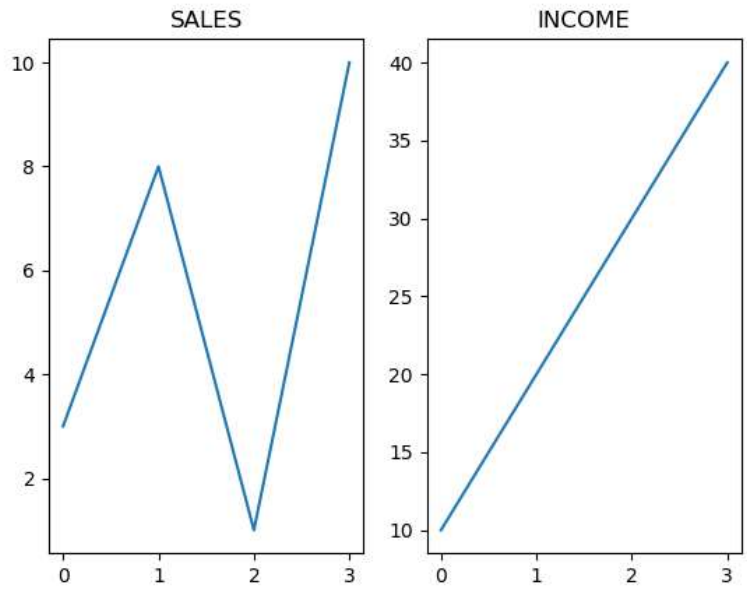
Example

2 plots, with titles:

```
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.show()

Result:



Super Title

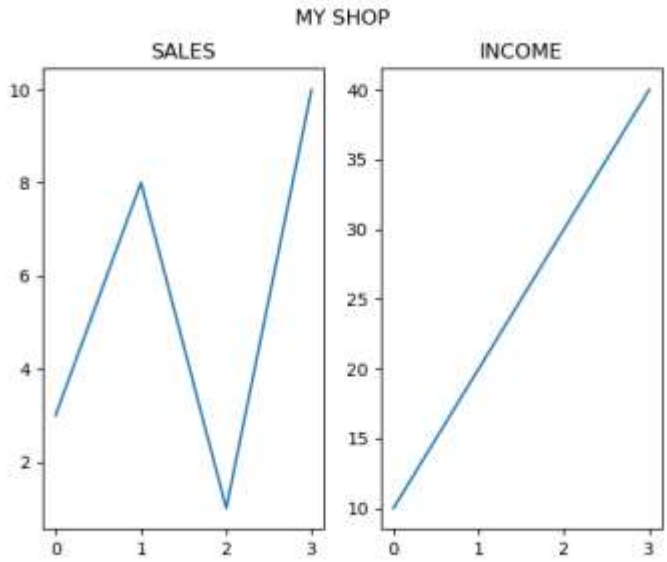
You can add a title to the entire figure with the suptitle() function:

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")
plt.show()
```

Result:



Matplotlib Scatter

Creating Scatter Plots

With Pyplot, you can use the scatter() function to draw a scatter plot.

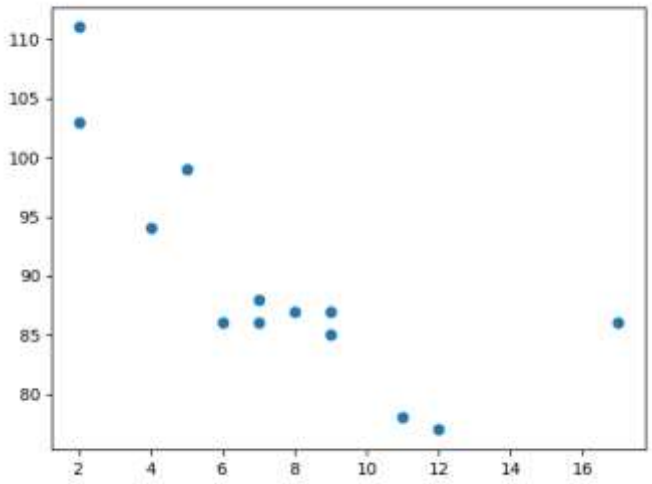
The scatter() function plots one dot for each observation. It needs two arrays of the same length, one for the values of the x-axis, and one for values on the y-axis:

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()
```

Result:



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.

Compare Plots

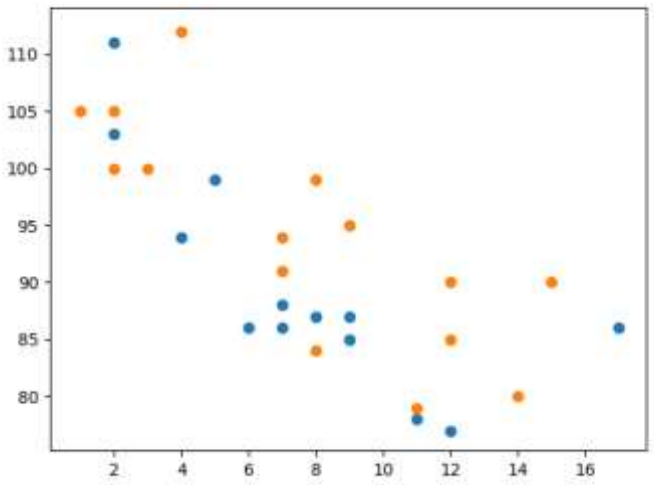
In the example above, there seems to be a relationship between speed and age, but what if we plot the observations from another day as well? Will the scatter plot tell us something else?

Example

Draw two plots on the same figure:

```
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)

#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)

plt.show()
```

Result:



Note: The two plots are plotted with two different colors, by default blue and orange, you will learn how to change colors later in this chapter.
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.

Colors

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

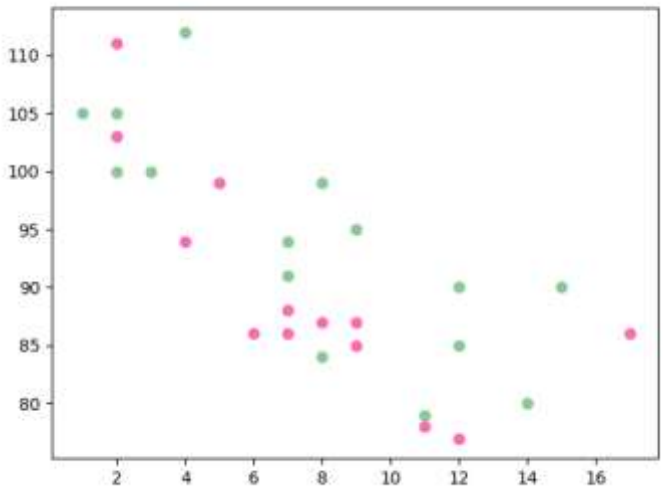
Example

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()
```

Result:



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:

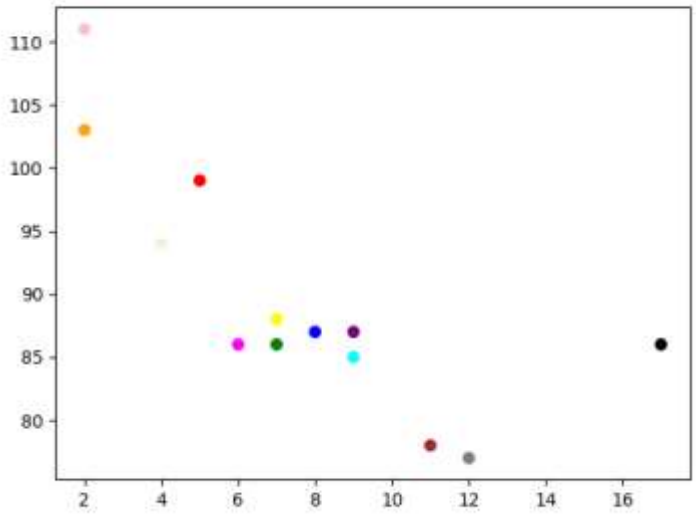
Note: You *cannot* use the color argument for this, only the c argument.

Example

Set your own color of the markers:

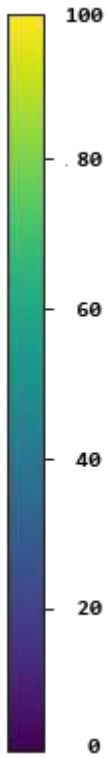
```
import matplotlib.pyplot as plt
import numpy as np

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","brown","gray"
,"cyan","magenta"])

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

plt.show()
```

Result:



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.

How to Use the ColorMap

You can specify the colormap with the keyword argument cmap with the value of the colormap, in this case 'viridis' which is one of the built-in colormaps available in Matplotlib.
In addition you have to create an array with values (from 0 to 100), one value for each of the point in the scatter plot:

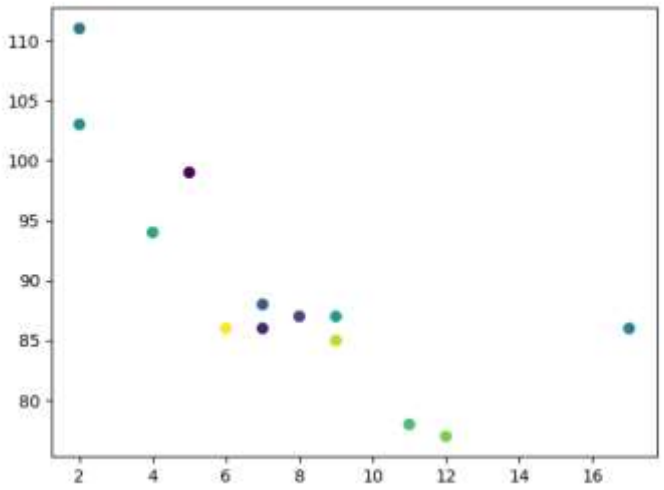
Example

Create a color array, and specify a colormap in the 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])
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()
```

Result:



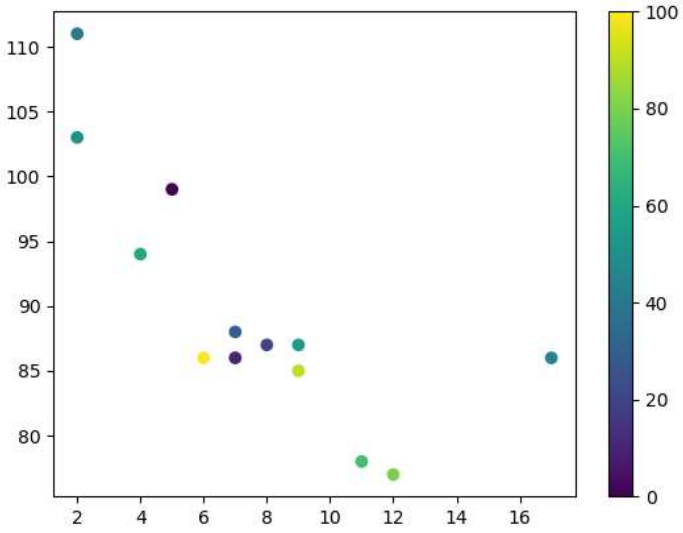
You can include the colormap in the drawing by including the plt.colorbar() statement:

Example

Include the actual colormap:

```
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([0, 10, 20, 30, 40, 45, 50, 55, 60, 70, 80, 90, 100])

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

plt.colorbar()

plt.show()
```

Result:



Available ColorMaps

You can choose any of the built-in colormaps:

Name	Reverse
Accent	Accent_r
Blues	Blues_r
BrBG	BrBG_r
BuGn	BuGn_r
BuPu	BuPu_r
CMRmap	CMRmap_r
Dark2	Dark2_r
GnBu	GnBu_r
Greens	Greens_r
Greys	Greys_r
OrRd	OrRd_r
Oranges	Oranges_r
PRGn	PRGn_r
Paired	Paired_r
Pastel1	Pastel1_r
Pastel2	Pastel2_r
PiYG	PiYG_r
PuBu	PuBu_r
PuBuGn	PuBuGn_r
PuOr	PuOr_r
PuRd	PuRd_r
Purples	Purples_r
RdBu	RdBu_r
RdGy	RdGy_r
RdPu	RdPu_r
RdYlBu	RdYlBu_r
RdYlGn	RdYlGn_r
Reds	Reds_r
Set1	Set1_r
Set2	Set2_r

Set3	Set3_r
Spectral	Spectral_r
Wistia	Wistia_r
YlGn	YlGn_r
YlGnBu	YlGnBu_r
YlOrBr	YlOrBr_r
YlOrRd	YlOrRd_r
afmhot	afmhot_r
autumn	autumn_r
binary	binary_r
bone	bone_r
brg	brg_r
bwr	bwr_r
cividis	cividis_r
cool	cool_r
coolwarm	coolwarm_r
copper	copper_r
cubehelix	cubehelix_r
flag	flag_r
gist_earth	gist_earth_r
gist_gray	gist_gray_r
gist_heat	gist_heat_r
gist_ncar	gist_ncar_r
gist_rainbow	gist_rainbow_r
gist_stern	gist_stern_r
gist_yarg	gist_yarg_r
gnuplot	gnuplot_r
gnuplot2	gnuplot2_r
gray	gray_r
hot	hot_r
hsv	hsv_r
inferno	inferno_r
jet	jet_r
magma	magma_r
nipy_spectral	nipy_spectral_r
ocean	ocean_r
pink	pink_r
plasma	plasma_r
prism	prism_r
rainbow	rainbow_r
seismic	seismic_r
spring	spring_r
summer	summer_r
tab10	tab10_r
tab20	tab20_r
tab20b	tab20b_r
tab20c	tab20c_r
terrain	terrain_r
twilight	twilight_r

twilight_shifted	twilight_shifted_r
viridis	viridis_r
winter	winter_r

Size

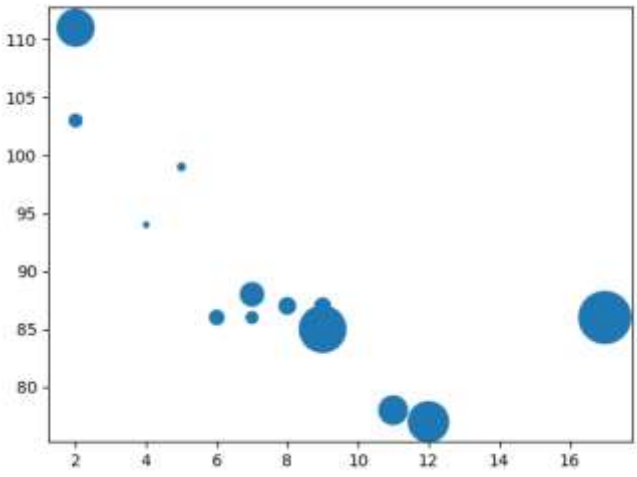
You can change the size of the dots with the s argument.
Just like colors, make sure the array for sizes has the same length as the arrays for the x- and y-axis:

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)

plt.show()
```

Result:



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:

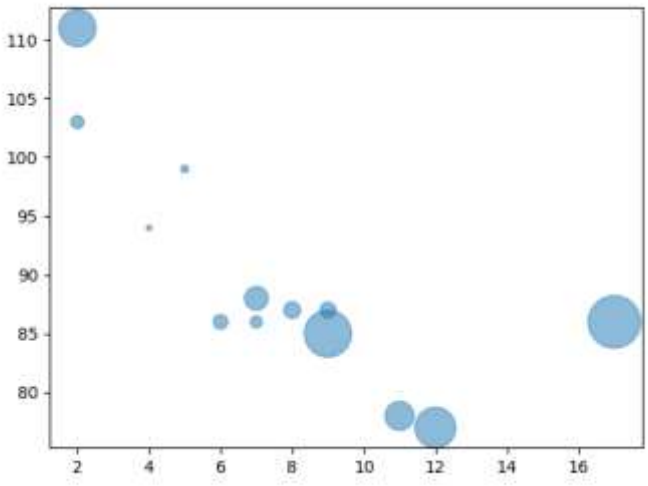
Example

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()
```

Result:



Combine Color Size and Alpha

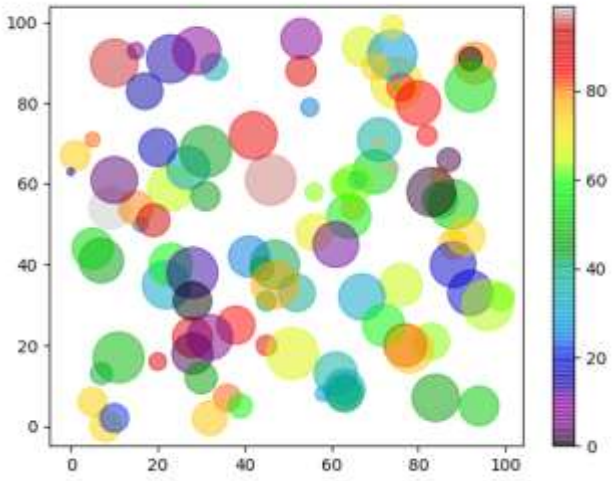
You can combine a colormap with different sizes on the dots. This is best visualized if the dots are transparent:

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))

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

plt.colorbar()

plt.show()
```

Result:



Matplotlib Bars

Creating Bars

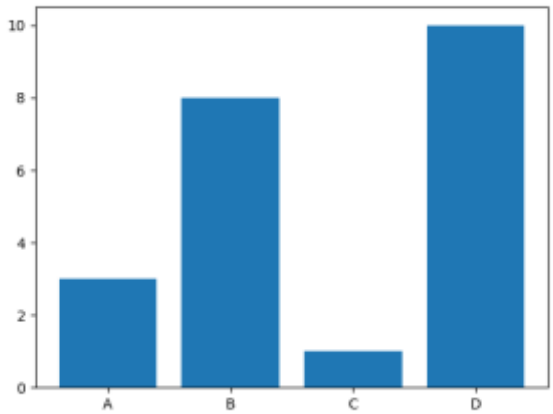
With Pyplot, you can use the bar() function to draw bar graphs:

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()
```

Result:



The bar() function takes arguments that describes the layout of the bars.

The categories and their values represented by the *first* and *second* argument as arrays.

Example

```
x = ["APPLES", "BANANAS"]
y = [400, 350]
plt.bar(x, y)
```

Horizontal Bars

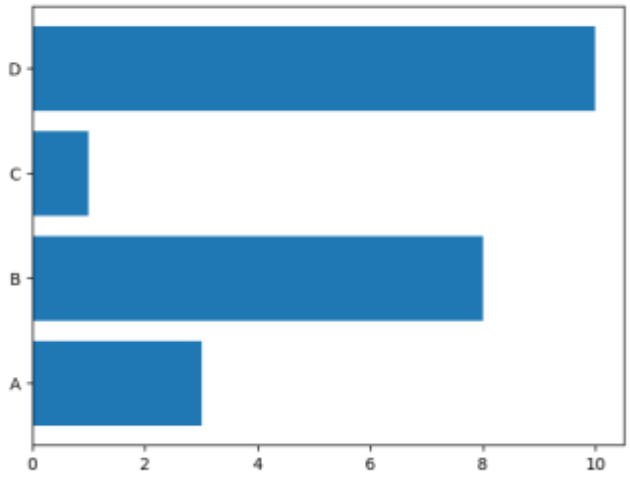
If you want the bars to be displayed horizontally instead of vertically, use the barh() function:

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()
```

Result:



Bar Color

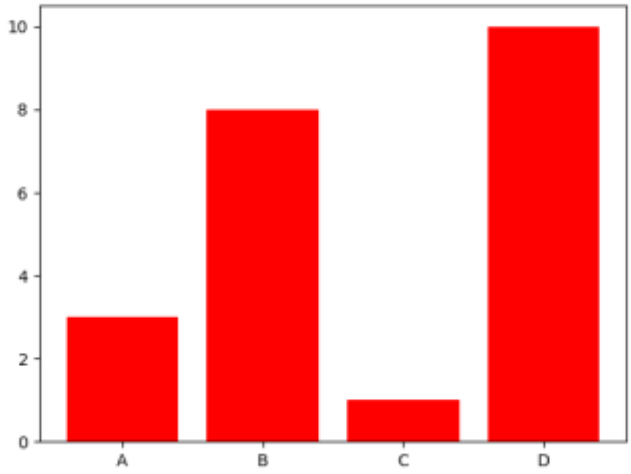
The bar() and barh() takes the keyword argument color to set the color of the bars:

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")
plt.show()
```

Result:



Color Names

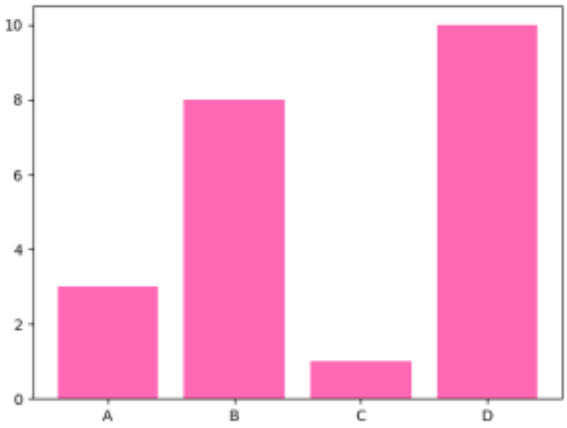
You can use any of the 140 supported color names.

Example

Draw 4 "hot pink" 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 = "hotpink")
plt.show()
```

Result:



Color Hex

Or you can use Hexadecimal color values:

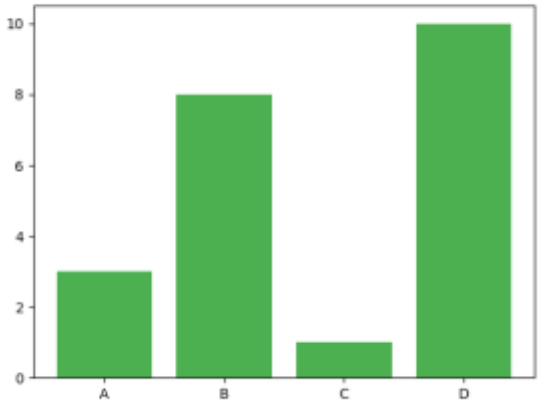
Example

Draw 4 bars with a beautiful green color:

```
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 = "#4CAF50")
plt.show()
```

Result:



Bar Width

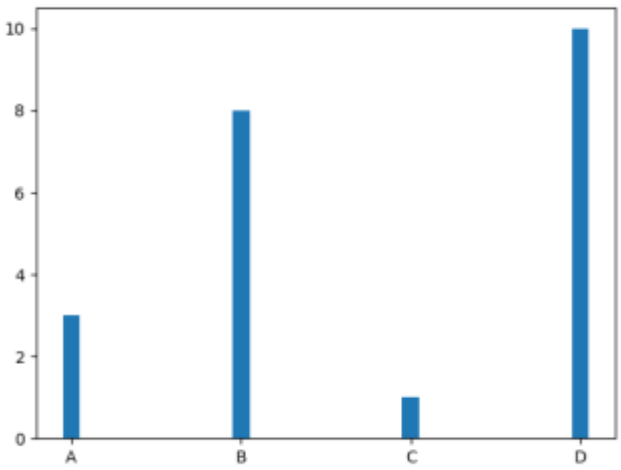
The bar() takes the keyword argument width to set the width of the bars:

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)
plt.show()
```

Result:



The default width value is 0.8

Note: For horizontal bars, use height instead of width.

Bar Height

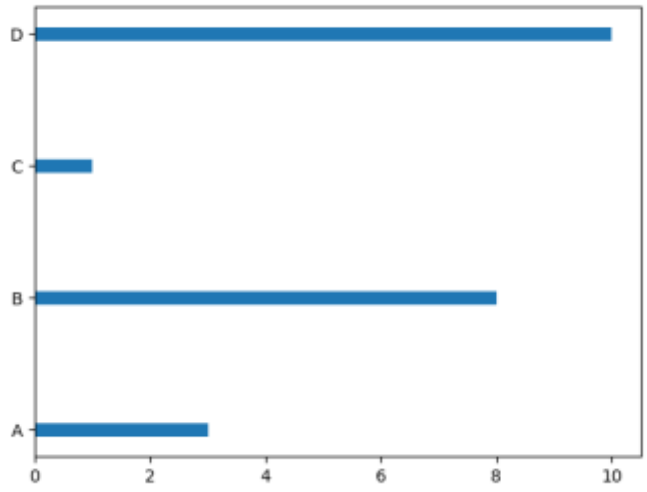
The barh() takes the keyword argument height to set the height of the bars:

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.barh(x, y, height = 0.1)
plt.show()
```

Result:



The default height value is 0.8

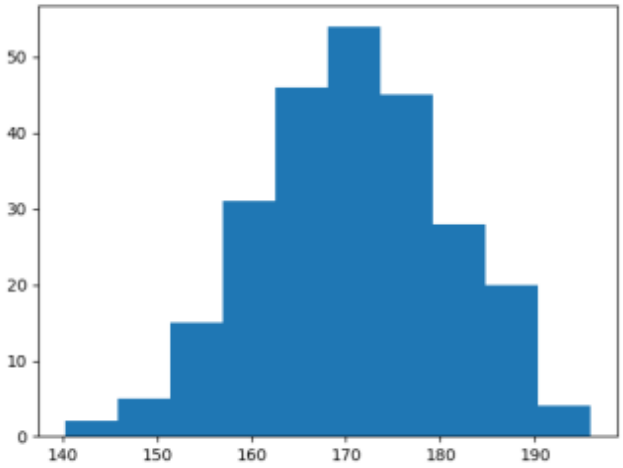
Matplotlib Histograms

Histogram

A histogram is a graph showing *frequency* distributions.

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

Example: Say you ask for the height of 250 people, you might end up with a histogram like this:



You can read from the histogram that there are approximately:

2 people from 140 to 145cm
5 people from 145 to 150cm
15 people from 151 to 156cm
31 people from 157 to 162cm
46 people from 163 to 168cm
53 people from 168 to 173cm
45 people from 173 to 178cm
28 people from 179 to 184cm
21 people from 185 to 190cm
4 people from 190 to 195cm

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.
For simplicity we use NumPy to randomly generate an array with 250 values, where the values will concentrate around 170, and the standard deviation is 10. Learn more about Normal Data Distribution in our Machine Learning Tutorial.

Example

A Normal Data Distribution by NumPy:

```
import numpy as np

x = np.random.normal(170, 10, 250)

print(x)
```

Result:

This will generate a *random* result, and could look like this:

```
[167.62255766 175.32495609 152.84661337 165.50264047 163.17457988
 162.29867872 172.83638413 168.67303667 164.57361342 180.81120541
 170.57782187 167.53075749 176.15356275 176.95378312 158.4125473
 187.8842668  159.03730075 166.69284332 160.73882029 152.22378865
 164.01255164 163.95288674 176.58146832 173.19849526 169.40206527
 166.88861903 149.90348576 148.39039643 177.90349066 166.72462233
 177.44776004 170.93335636 173.26312881 174.76534435 162.28791953
 166.77301551 160.53785202 170.67972019 159.11594186 165.36992993
 178.38979253 171.52158489 173.32636678 159.63894401 151.95735707
 175.71274153 165.00458544 164.80607211 177.50988211 149.28106703
 179.43586267 181.98365273 170.98196794 179.1093176  176.91855744
 168.32092784 162.33939782 165.18364866 160.52300507 174.14316386
 163.01947601 172.01767945 173.33491959 169.75842718 198.04834503
 192.82490521 164.54557943 206.36247244 165.47748898 195.26377975
 164.37569092 156.15175531 162.15564208 179.34100362 167.22138242
 147.23667125 162.86940215 167.84986671 172.99302505 166.77279814
 196.6137667  159.79012341 166.5840824  170.68645637 165.62204521
 174.5559345  165.0079216  187.92545129 166.86186393 179.78383824
 161.0973573  167.44890343 157.38075812 151.35412246 171.3107829
 162.57149341 182.49985133 163.24700057 168.72639903 169.05309467
 167.19232875 161.06405208 176.87667712 165.48750185 179.68799986
 158.7913483  170.22465411 182.66432721 173.5675715  176.85646836
```

157.31299754 174.88959677 183.78323508 174.36814558 182.55474697
180.03359793 180.53094948 161.09560099 172.29179934 161.22665588
171.88382477 159.04626132 169.43886536 163.75793589 157.73710983
174.68921523 176.19843414 167.39315397 181.17128255 174.2674597
186.05053154 177.06516302 171.78523683 166.14875436 163.31607668
174.01429569 194.98819875 169.75129209 164.25748789 180.25773528
170.44784934 157.81966006 171.33315907 174.71390637 160.55423274
163.92896899 177.29159542 168.30674234 165.42853878 176.46256226
162.61719142 166.60810831 165.83648812 184.83238352 188.99833856
161.3054697 175.30396693 175.28109026 171.54765201 162.08762813
164.53011089 189.86213299 170.83784593 163.25869004 198.68079225
166.95154328 152.03381334 152.25444225 149.75522816 161.79200594
162.13535052 183.37298831 165.40405341 155.59224806 172.68678385
179.35359654 174.19668349 163.46176882 168.26621173 162.97527574
192.80170974 151.29673582 178.65251432 163.17266558 165.11172588
183.11107905 169.69556831 166.35149789 178.74419135 166.28562032
169.96465166 178.24368042 175.3035525 170.16496554 158.80682882
187.10006553 178.90542991 171.65790645 183.19289193 168.17446717
155.84544031 177.96091745 186.28887898 187.89867406 163.26716924
169.71242393 152.9410412 158.68101969 171.12655559 178.1482624
187.45272185 173.02872935 163.8047623 169.95676819 179.36887054
157.01955088 185.58143864 170.19037101 157.221245   168.90639755
178.7045601 168.64074373 172.37416382 165.61890535 163.40873027
168.98683006 149.48186389 172.20815568 172.82947206 173.71584064
189.42642762 172.79575803 177.00005573 169.24498561 171.55576698
161.36400372 176.47928342 163.02642822 165.09656415 186.70951892
153.27990317 165.59289527 180.34566865 189.19506385 183.10723435
173.48070474 170.28701875 157.24642079 157.9096498 176.4248199]

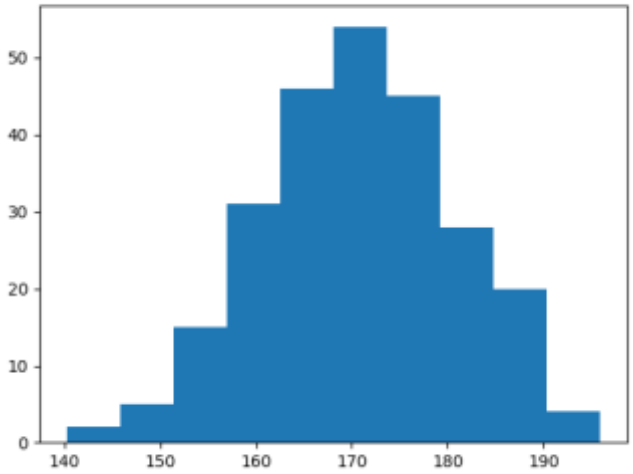
The hist() function will read the array and produce a histogram:

Example

A simple histogram:

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

x = np.random.normal(170, 10, 250)

plt.hist(x)
plt.show()
```

Result:



Matplotlib Pie Charts

Creating Pie Charts

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

Example

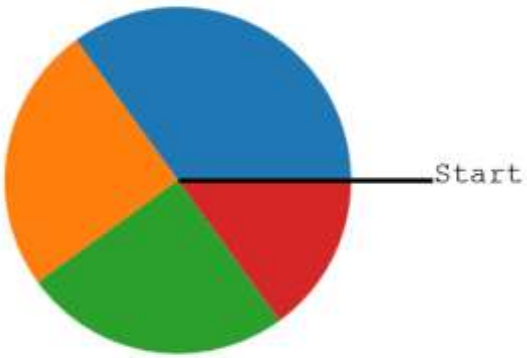
A simple pie chart:

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

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

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

Result:

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*:



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)

Labels

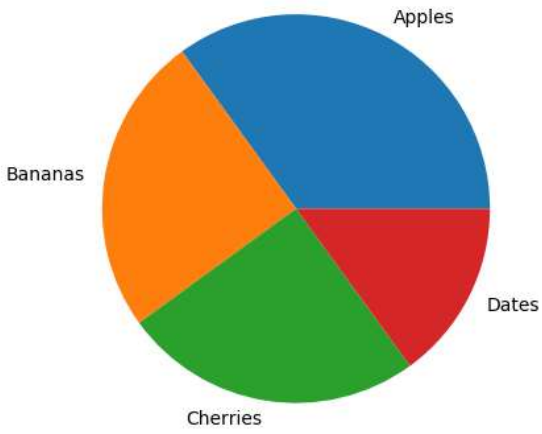
Add labels to the pie chart with the label parameter.
The label parameter must be an array with one label for each wedge:

Example

A simple pie chart:

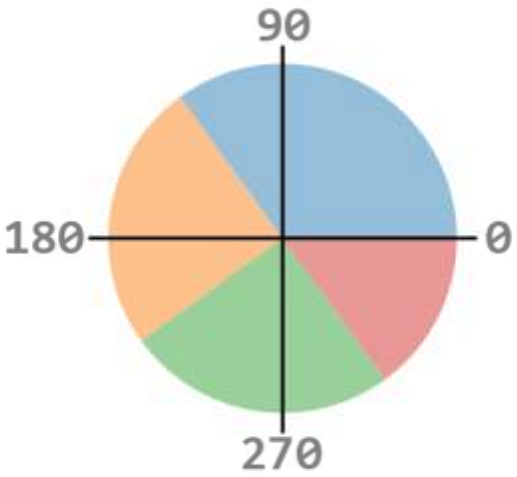
```
import matplotlib.pyplot as plt
import numpy as np

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

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

Result:



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:

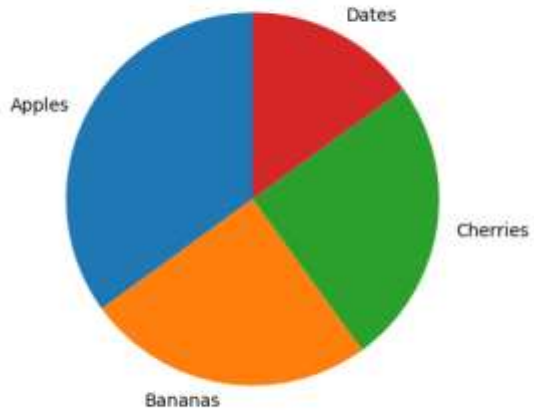


Example

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()
```

Result:



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.

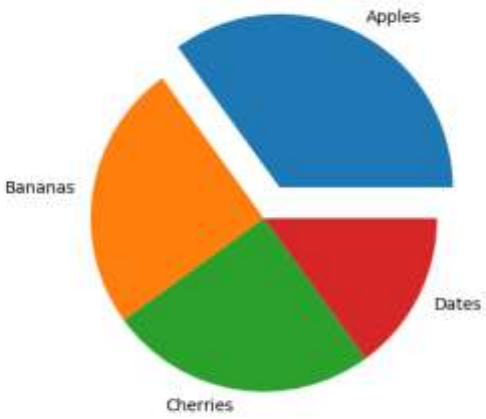
Each value represents how far from the center each wedge is displayed:

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]

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

Result:



Shadow

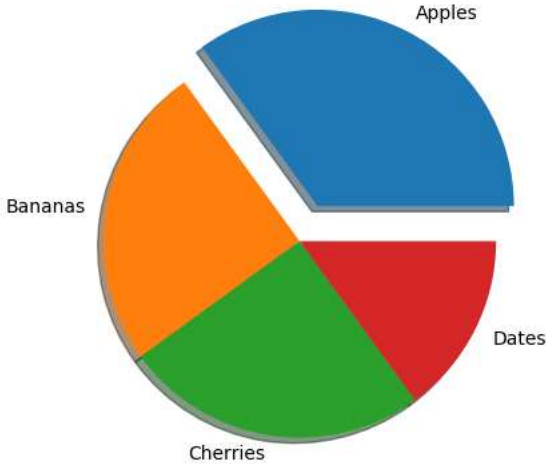
Add a shadow to the pie chart by setting the shadows parameter to True:

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)
plt.show()
```

Result:



Colors

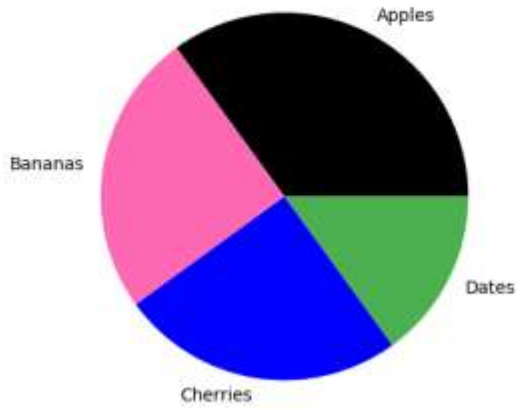
You can set the color of each wedge with the colors parameter.
The colors parameter, if specified, must be an array with one value for each wedge:

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"]

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

Result:



You can use Hexadecimal color values, any of the 140 supported color names, or one of these shortcuts:
'r' - Red
'g' - Green
'b' - Blue
'c' - Cyan
'm' - Magenta
'y' - Yellow
'k' - Black
'w' - White

Legend

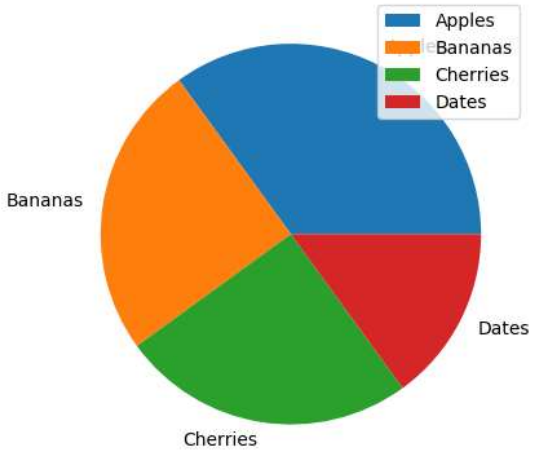
To add a list of explanation for each wedge, use the legend() function:

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()
plt.show()
```

Result:



Legend With Header

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

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()
```

Result: