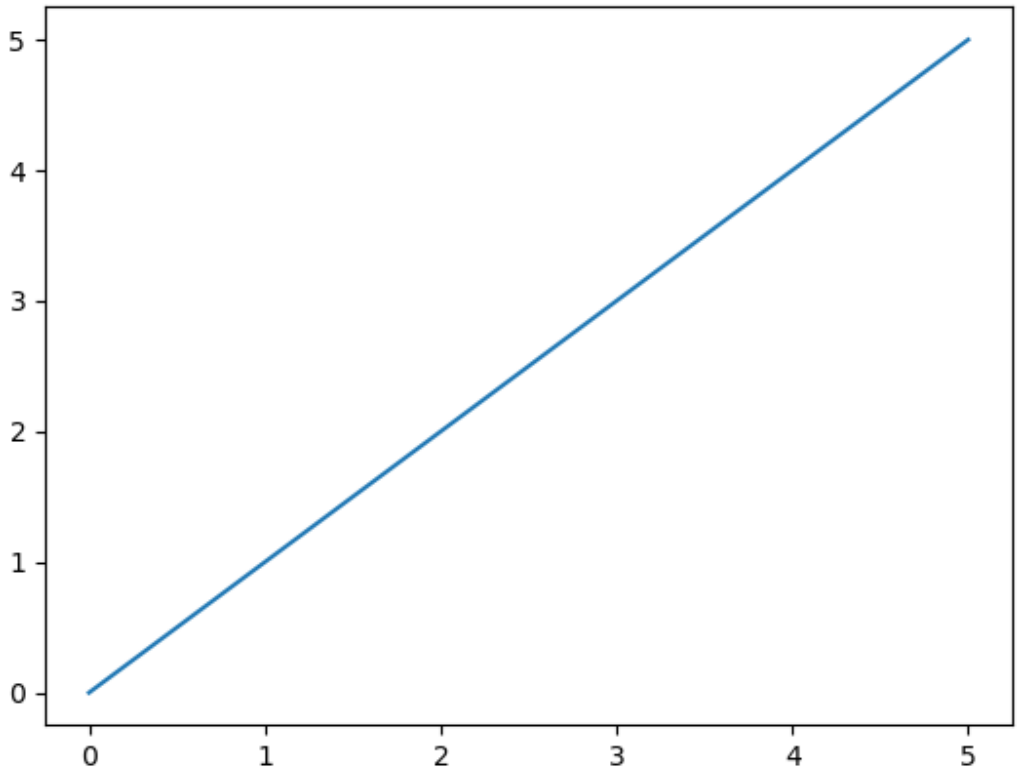
# Matplotlib Tutorial




## 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

```
import matplotlib

print(matplotlib.__version__)
```

**Note:** two underscore characters are used in `__version__`.

# Pyplot

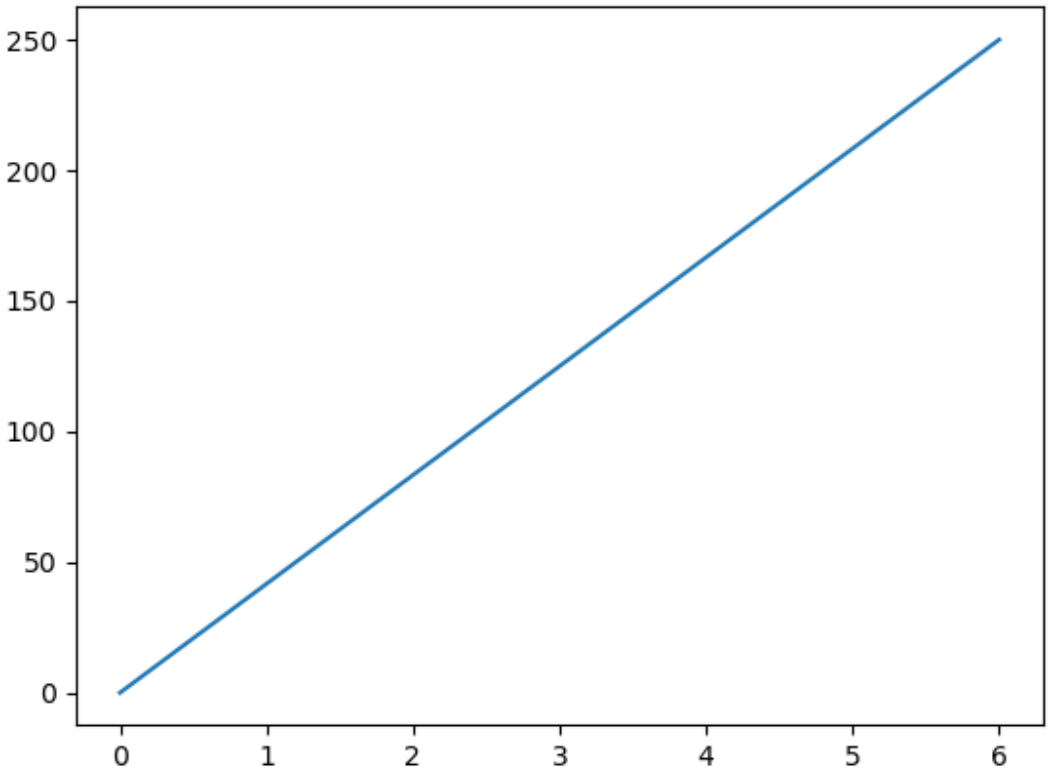
Most of the Matplotlib utilities lies under the `pyplot` submodule, and are usually imported under the `plt` alias:

```
import matplotlib.pyplot 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()
```




You will learn more about drawing (plotting) in the next chapters.

## 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()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()
```

# Plotting Without Line

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

## Example

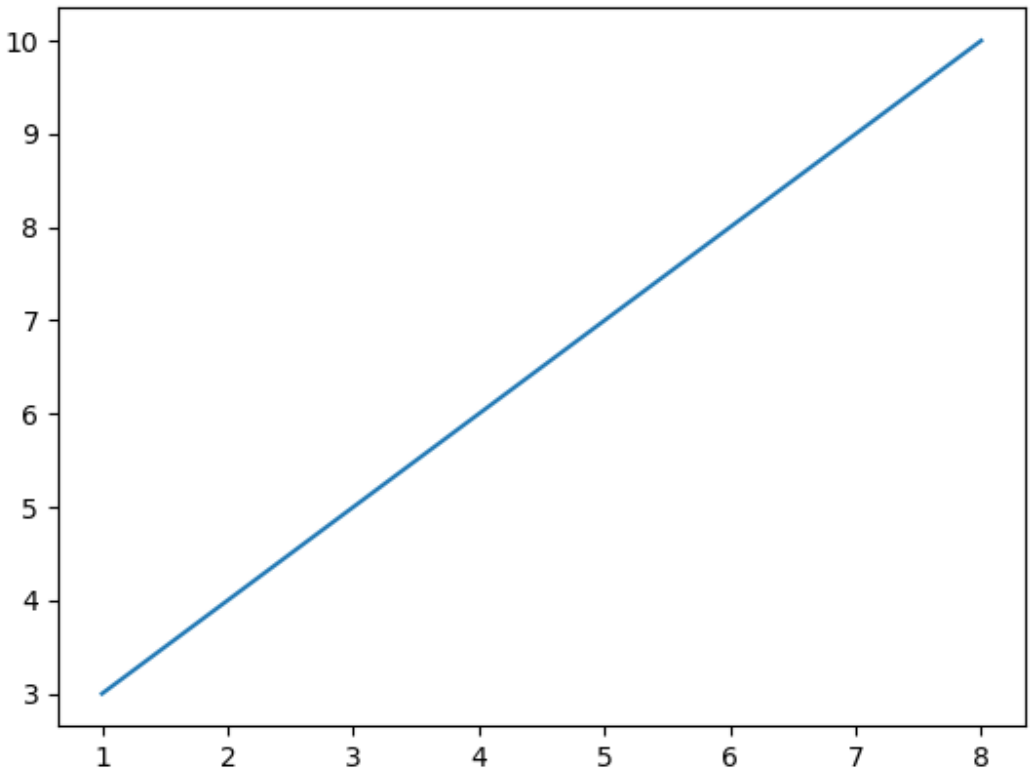
Draw two points in the diagram, one at position (1, 3) and one in position (8, 10):

## 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()
```




**Multiple Points**

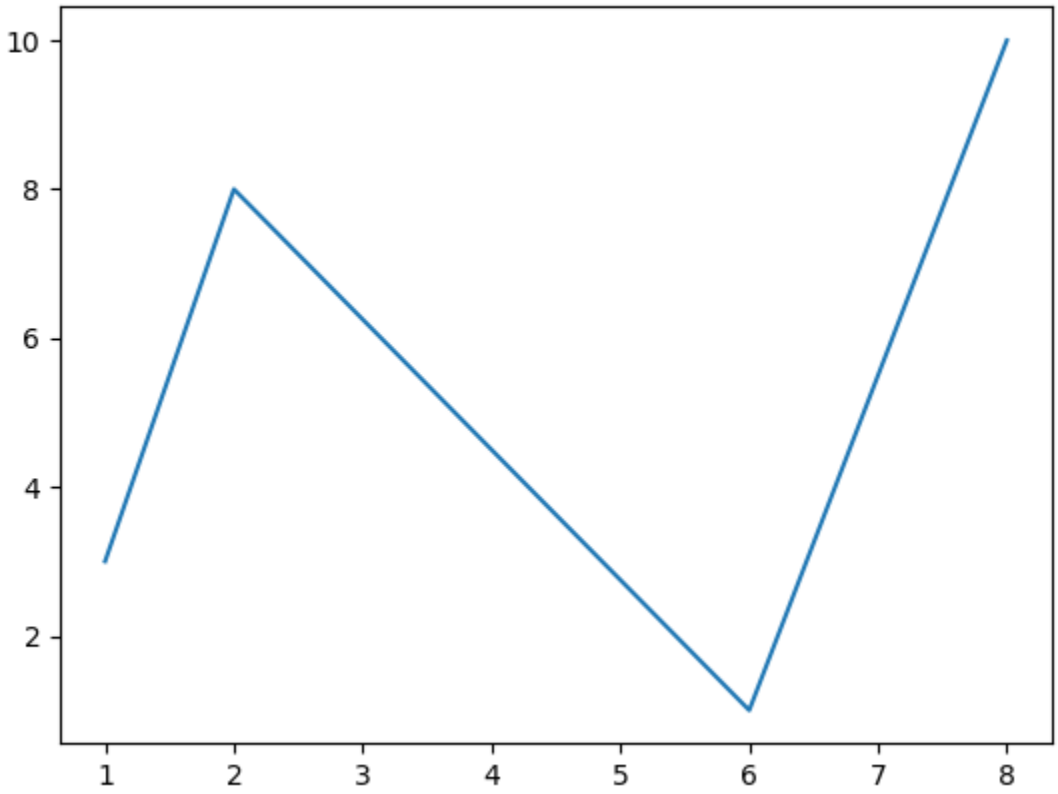
You can plot as many points as you like, just make sure you have the same number of points in both axis.

## 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()
```




# Default X-Points

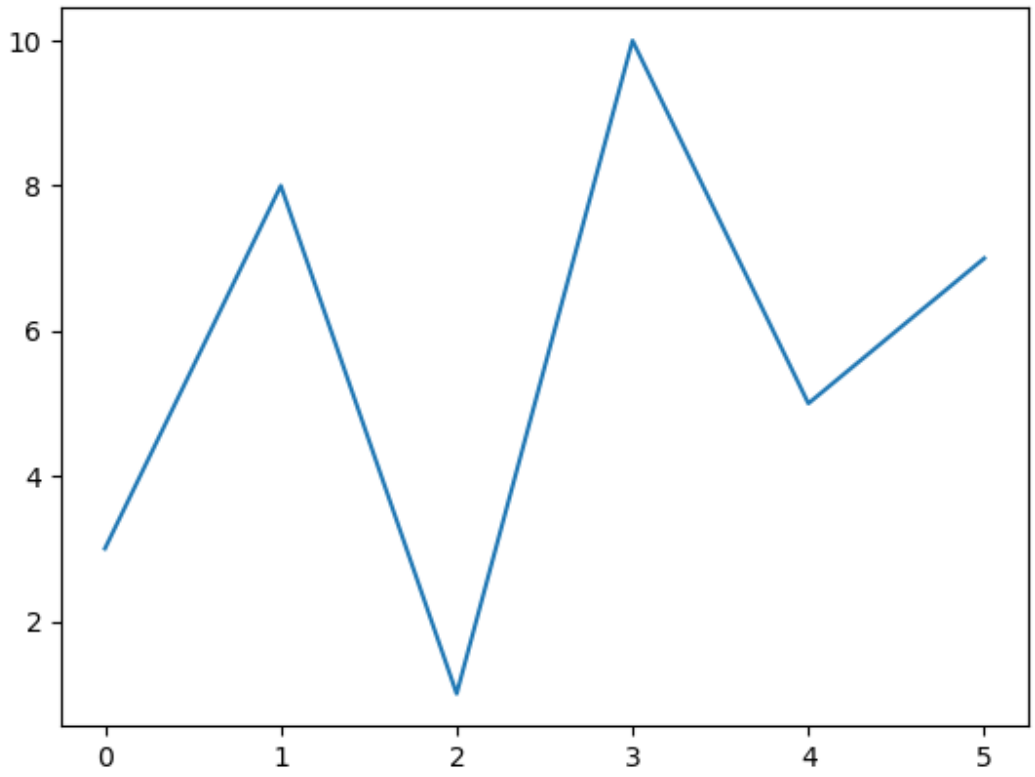
If we do not specify the points on the x-axis, they will get the default values 0, 1, 2, 3 etc., depending on the length of the y-points.

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()
```




# Matplotlib Markers

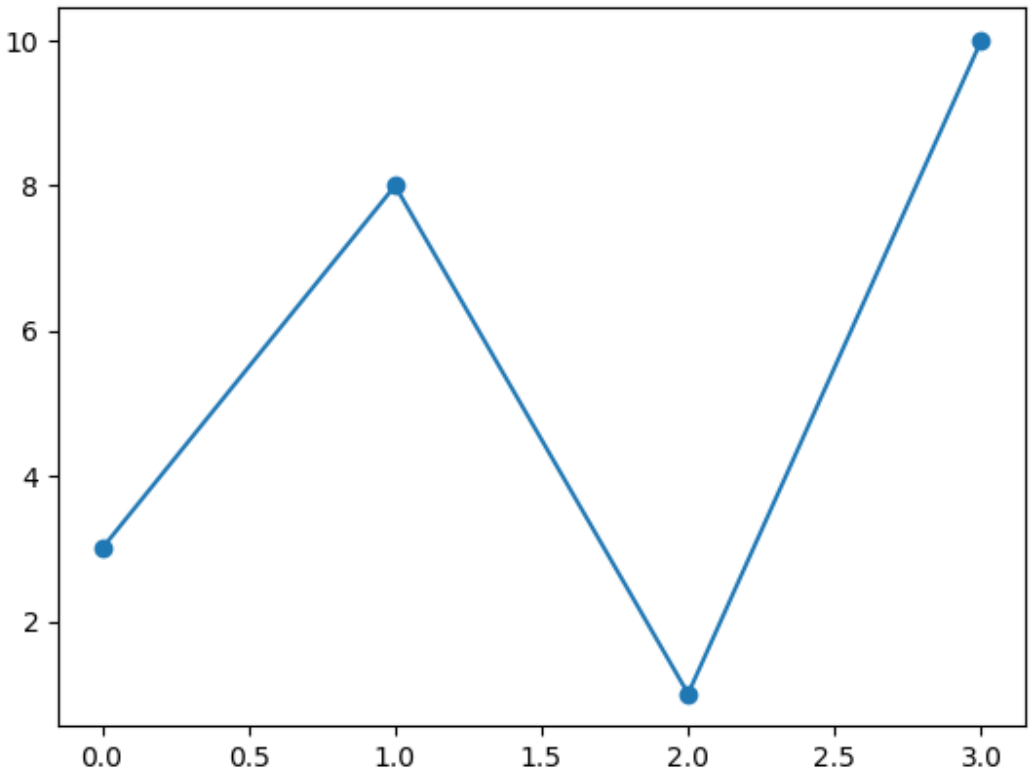
## Markers

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

## Example

Mark each point with a circle:

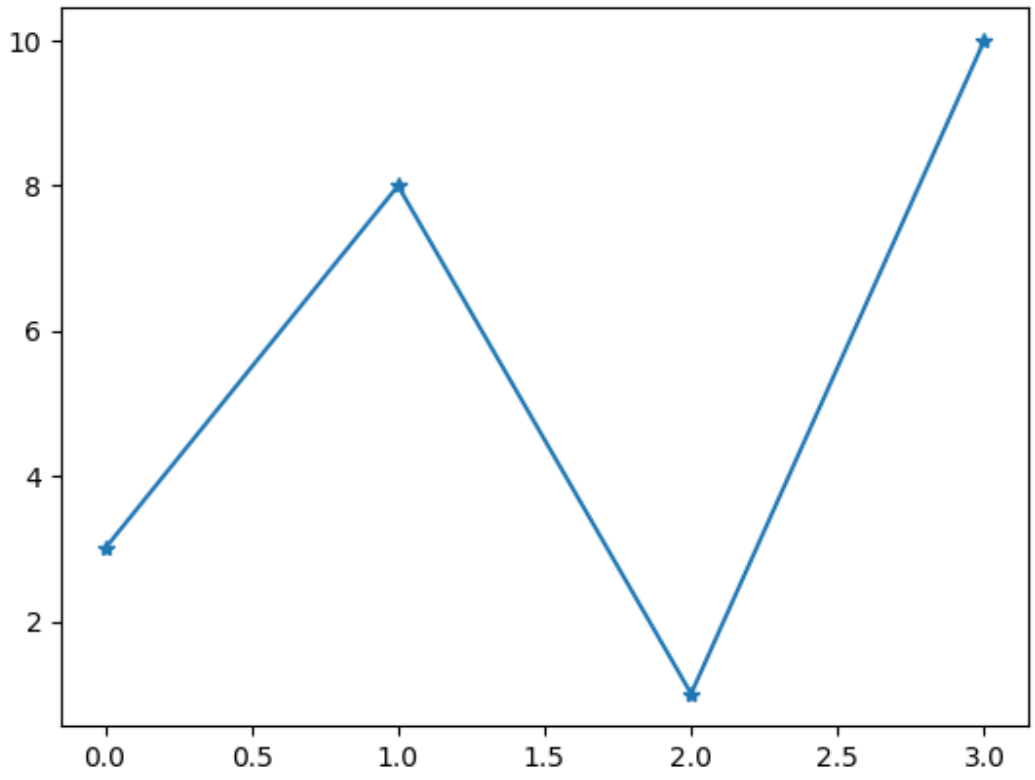
```
import matplotlib.pyplot as plt
import numpy as np

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

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




## Example

Mark each point with a star:

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




| 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 also use the *shortcut string notation* parameter to specify the marker.

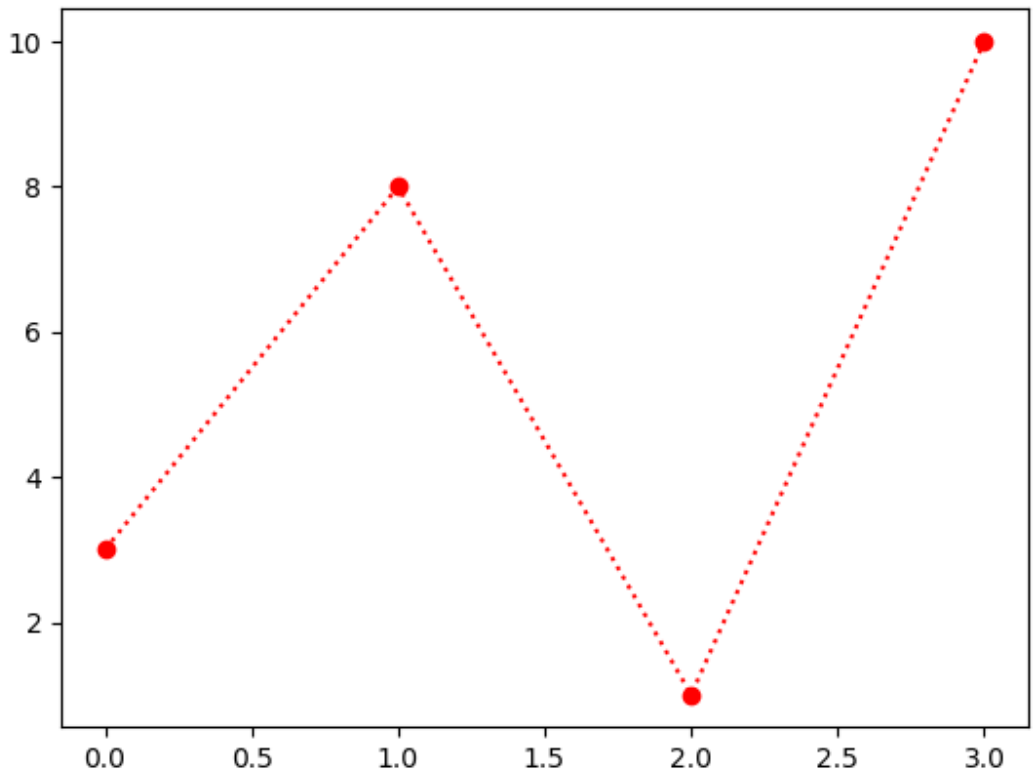
This parameter is also called `fmt`, and is written with this syntax:

*`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()
```




**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 |

# Color Reference

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

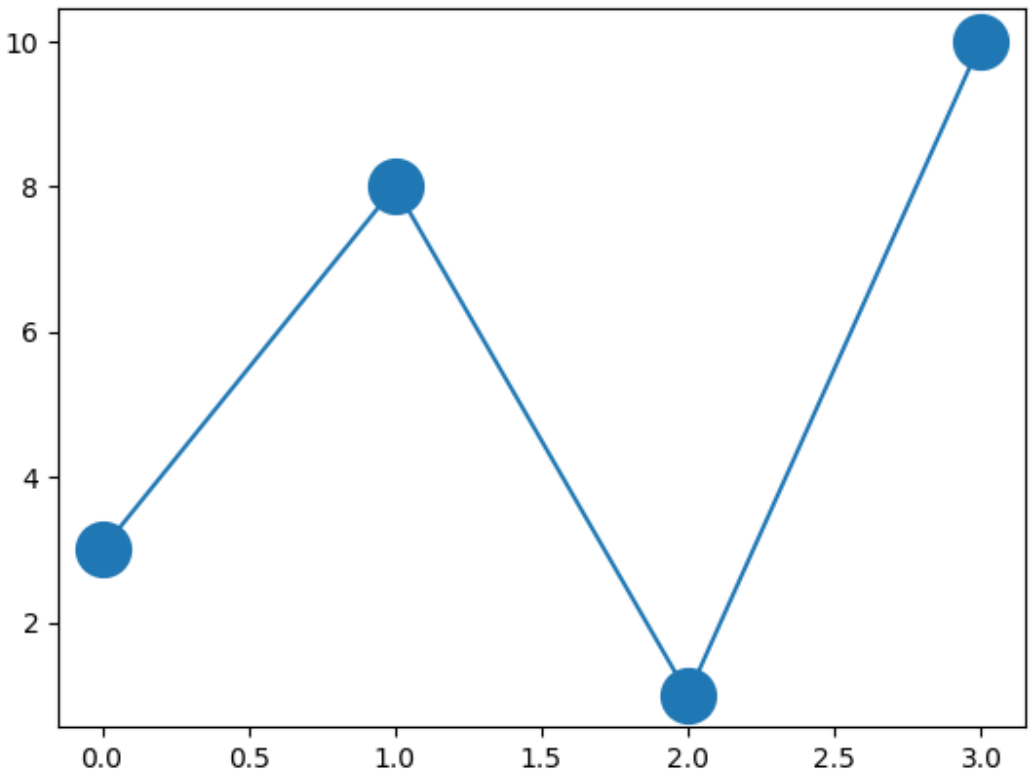
# Marker Size

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()
```




# 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:

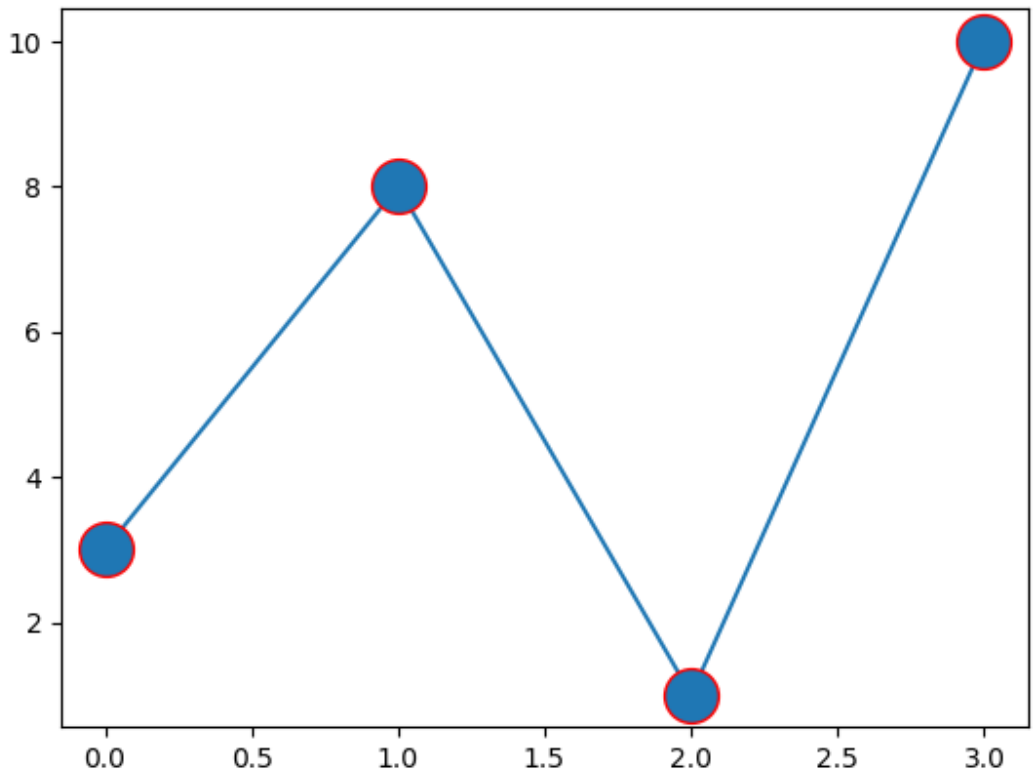
```
import matplotlib.pyplot as plt
import numpy as np

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

plt.plot(ypoints, marker = 'o', ms = 20, mec = 'r')
plt.show()
```

**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()
```

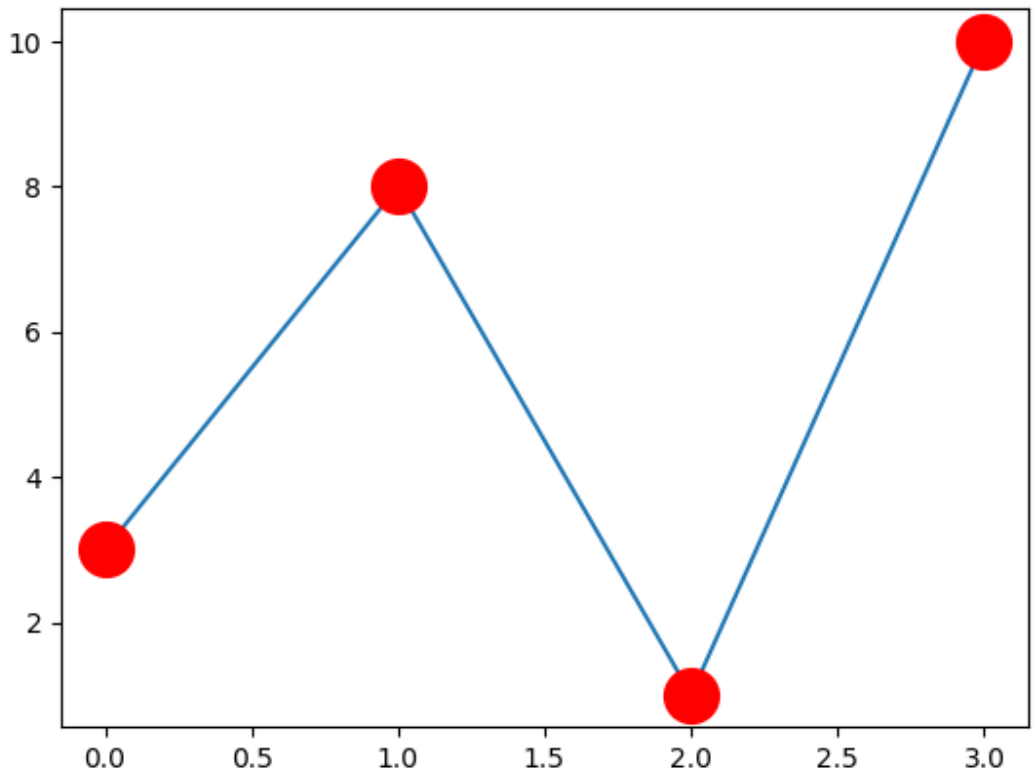



Use *both* the `mec` and `mfc` arguments to color the entire marker:

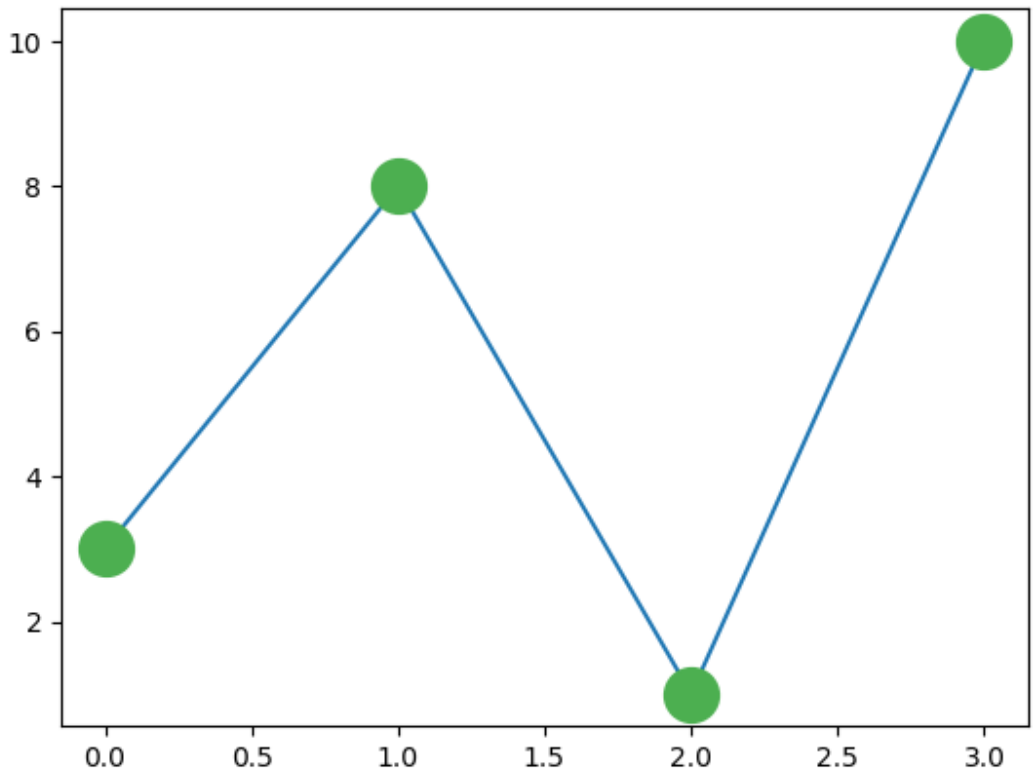
## Example

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()
```




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

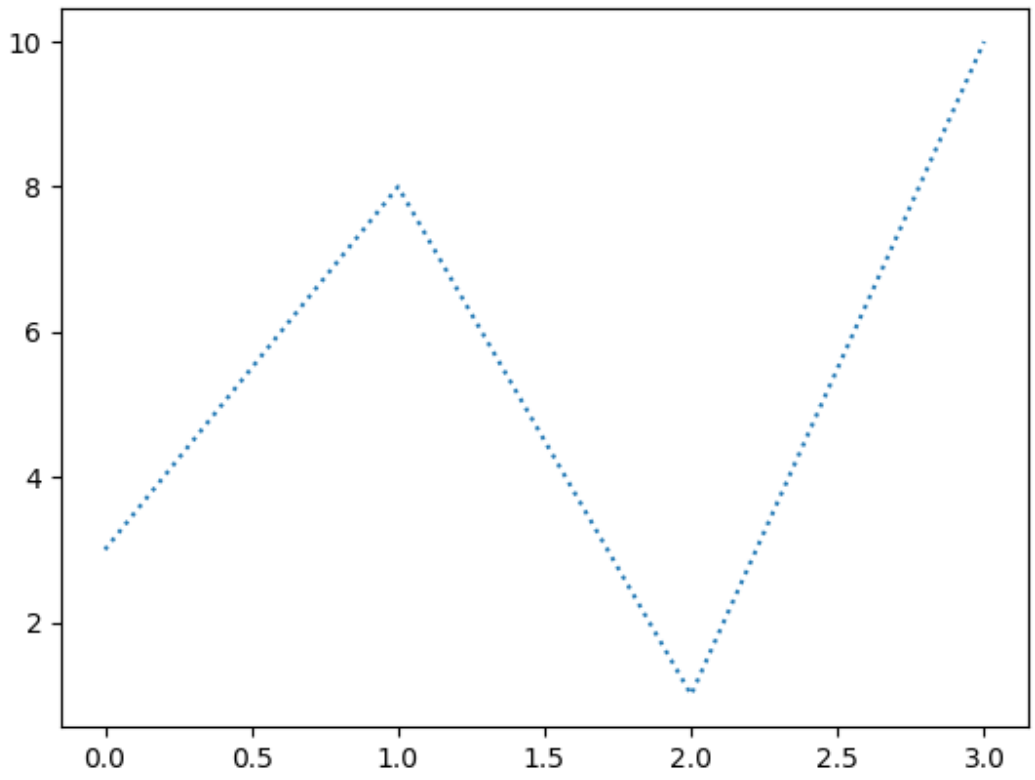



```
      plt.plot(ypoints, marker = 'o', ms = 20, mec = 'hotpink', mfc
= 'hotpink')
```

# Linestyle

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

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

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

plt.plot(ypoints, linestyle = 'dotted')
plt.show()
```




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

The line style can be written in a shorter syntax:

`linestyle` can be written as `ls`.

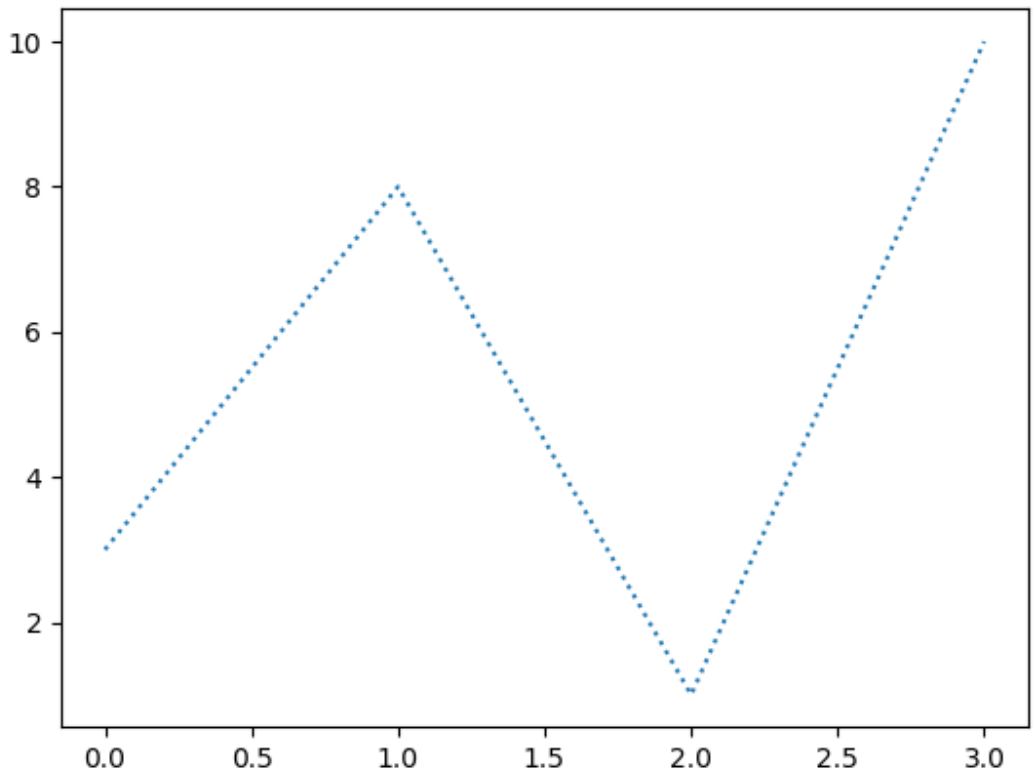
`dotted` can be written as `:`.

`dashed` can be written as `--`.

## Example

Shorter syntax:

```
plt.plot(ypoints, ls = ':')
```




# Line Styles

You can choose any of these styles:

| Style | Or |
| --- | --- |
| 'solid' (default) | '-' |
| 'dotted' | ':' |

| | |
|---|---|
| 'dashed' | '--' |
| 'dashdot' | '-.' |
| 'None' | '' or ' ' |

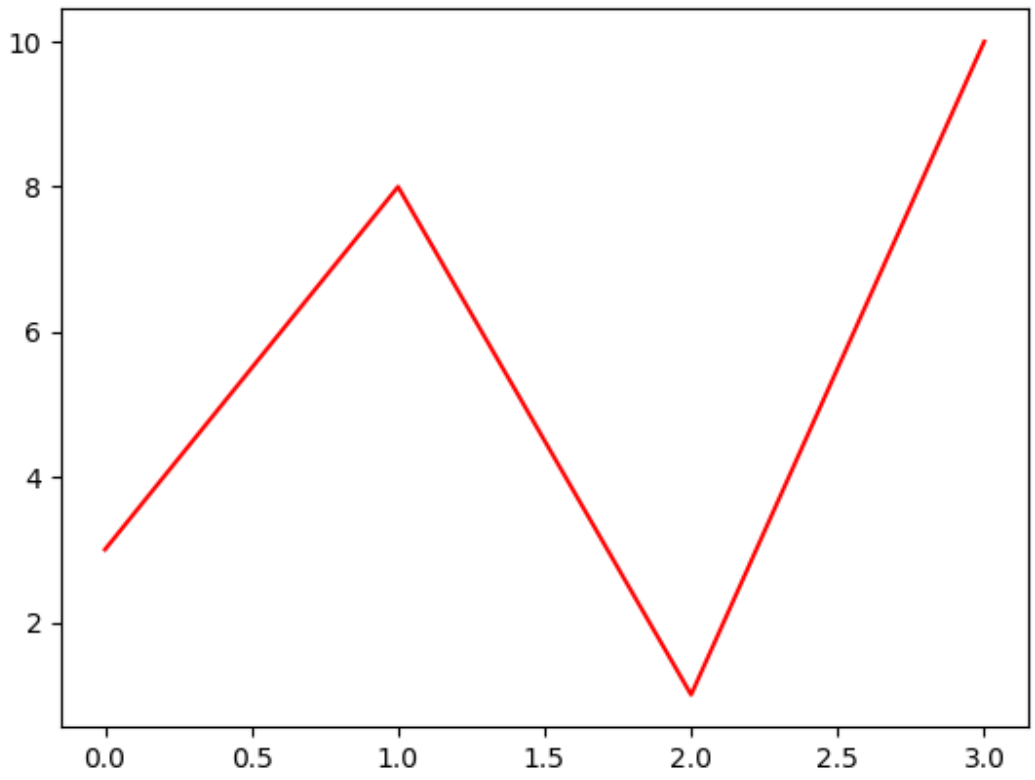
# Line Color

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()
```




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

# Line Width

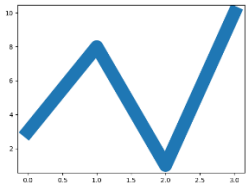
You can use the keyword argument `linewidth` or the shorter `lw` to change the width of the line.

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()
```

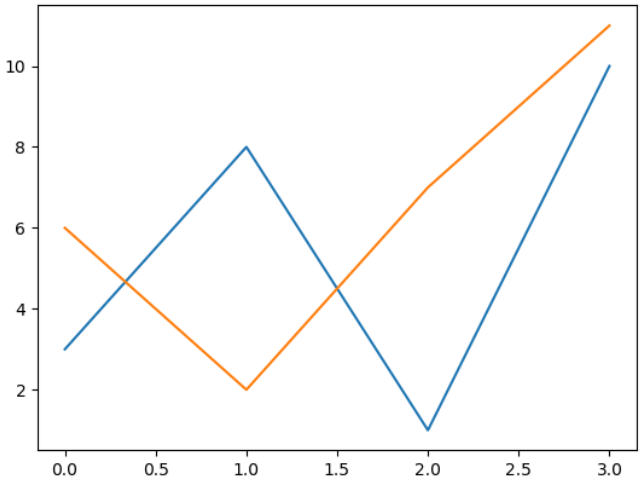
# Multiple Lines

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()
```



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).)

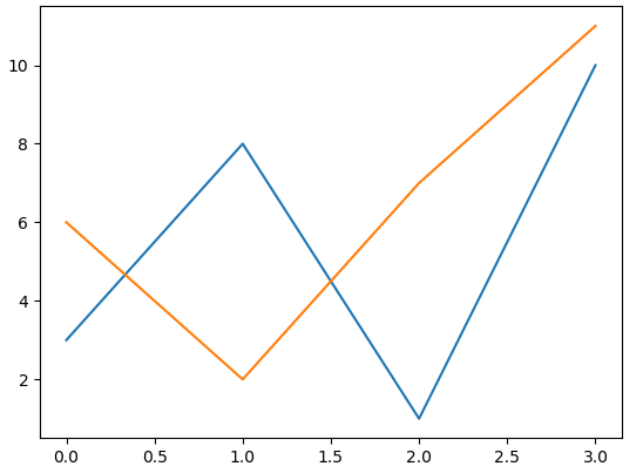
The x- and y- values come in pairs:

## 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:

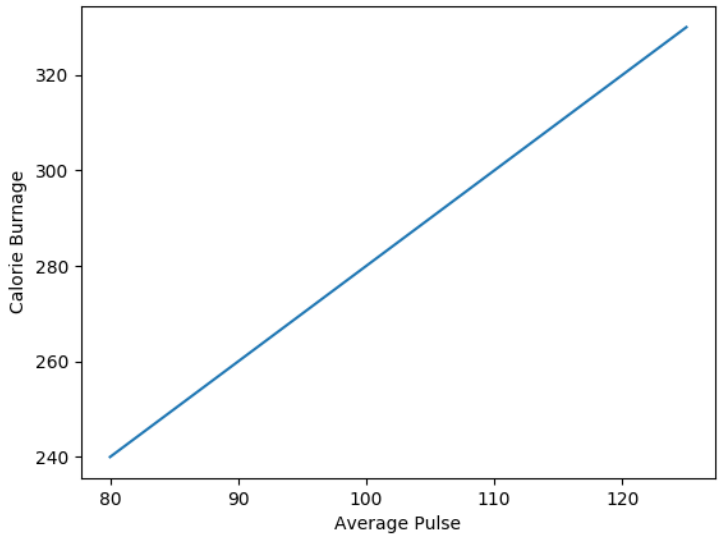



# Create Labels for a Plot

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()
```



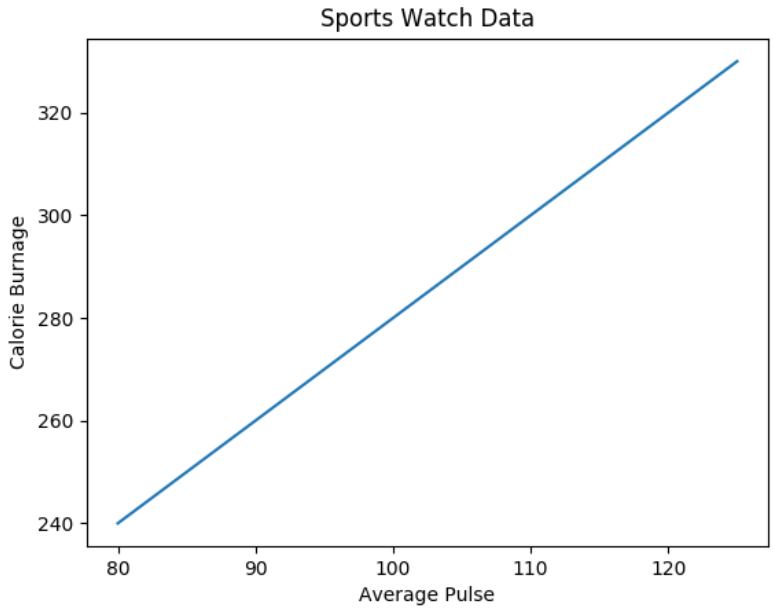

# Create a Title for a Plot

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()
```




# Set Font Properties for Title and Labels

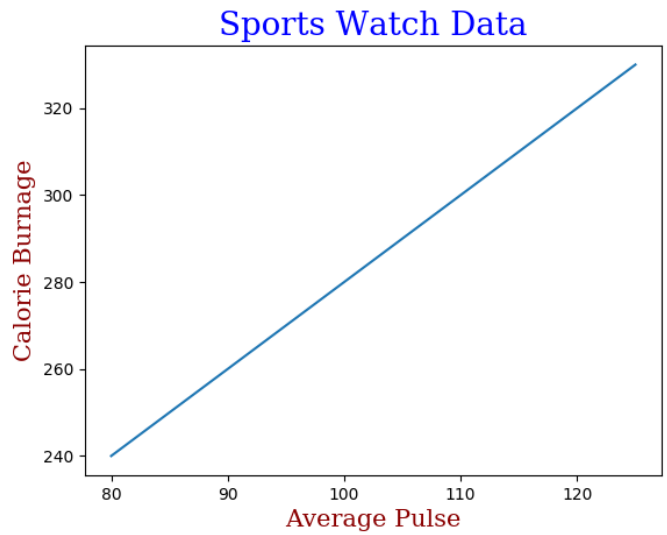
You can use the `fontdict` parameter in `xlabel()`, `ylabel()`, and `title()` to set font properties for the title and labels.

## 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()
```




# 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:

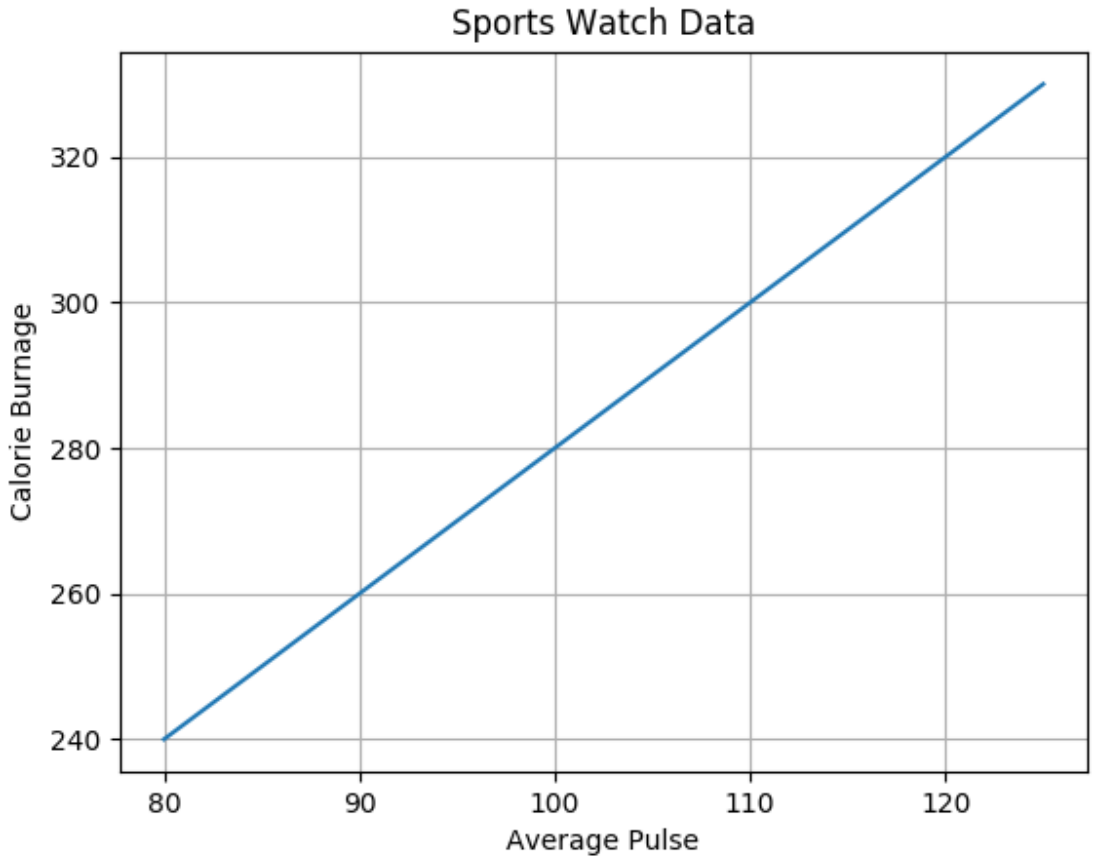
```
import numpy as np
import matplotlib.pyplot as plt

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()
```


# Add Grid Lines to a Plot

With Pyplot, you can use the `grid()` function to 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()
```




# Specify Which Grid Lines to Display

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

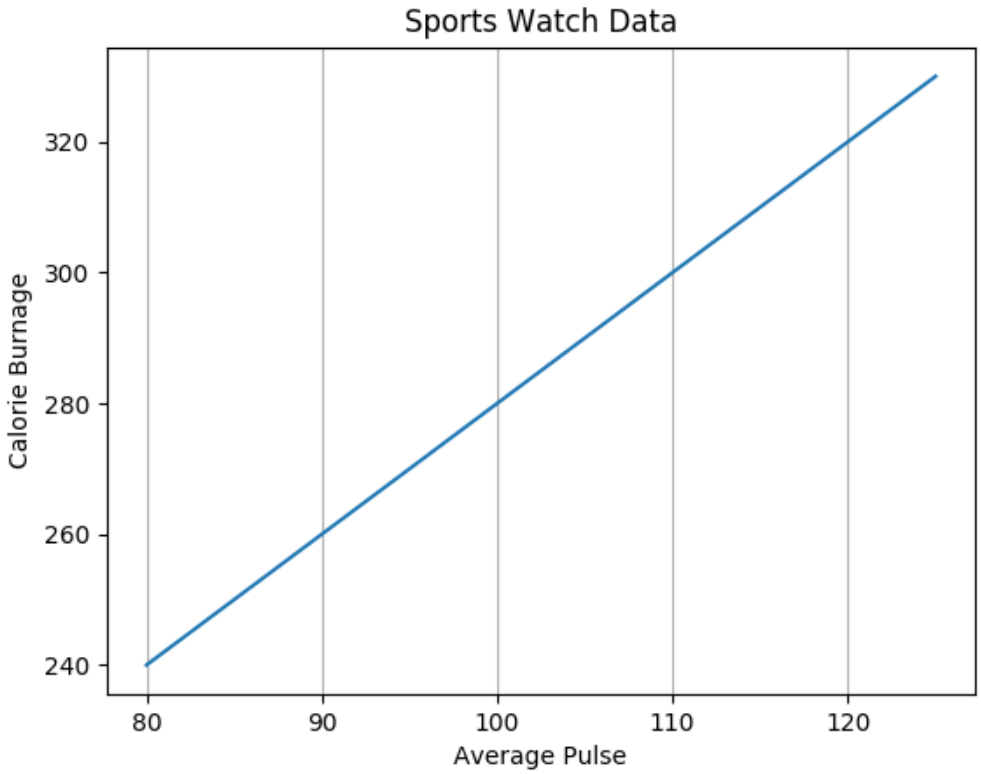
Legal values are: 'x', 'y', and 'both'. Default value is 'both'.

## Example

Display only grid lines for the x-axis:

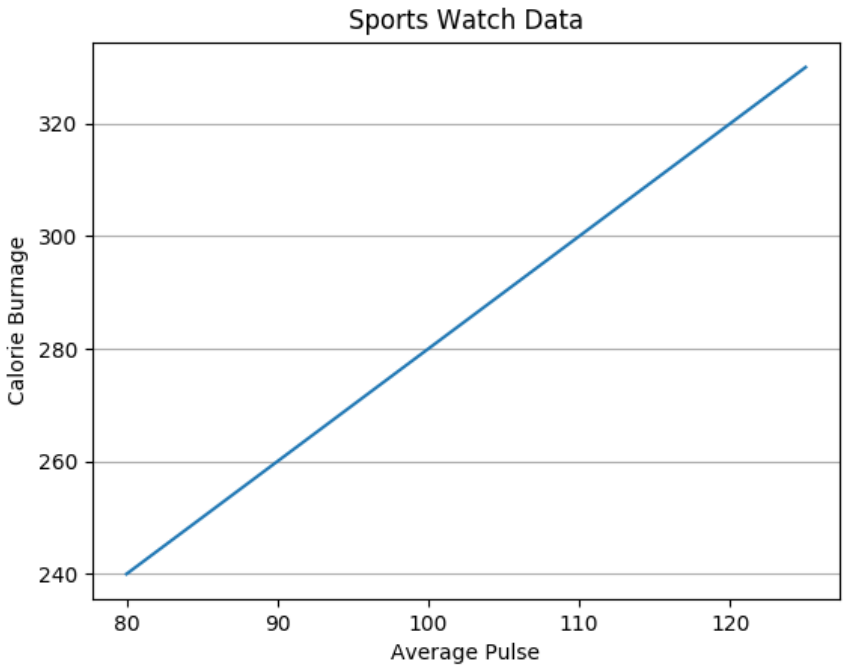
```
import numpy as np
import matplotlib.pyplot as plt

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()
```




```
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()
```




# Set Line Properties for the Grid

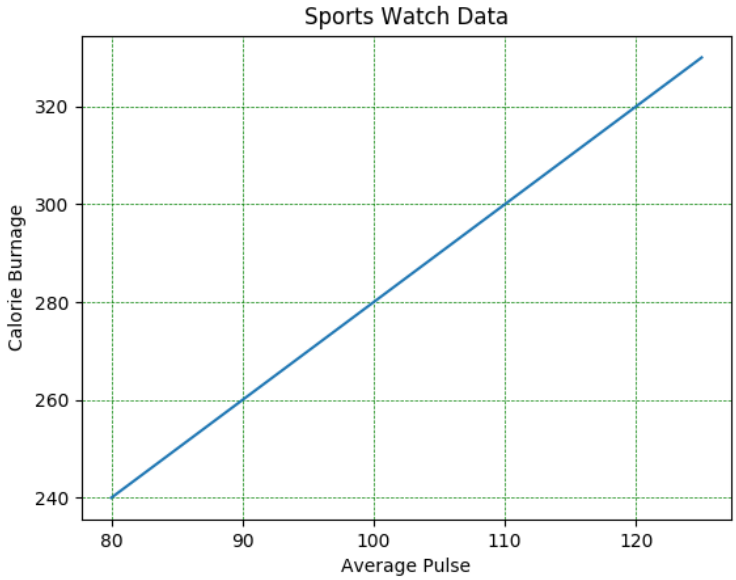
You can also set the line properties of the grid, like this: grid(color = '*color*', linestyle = '*linestyle*', linewidth = *number*).

## 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()
```

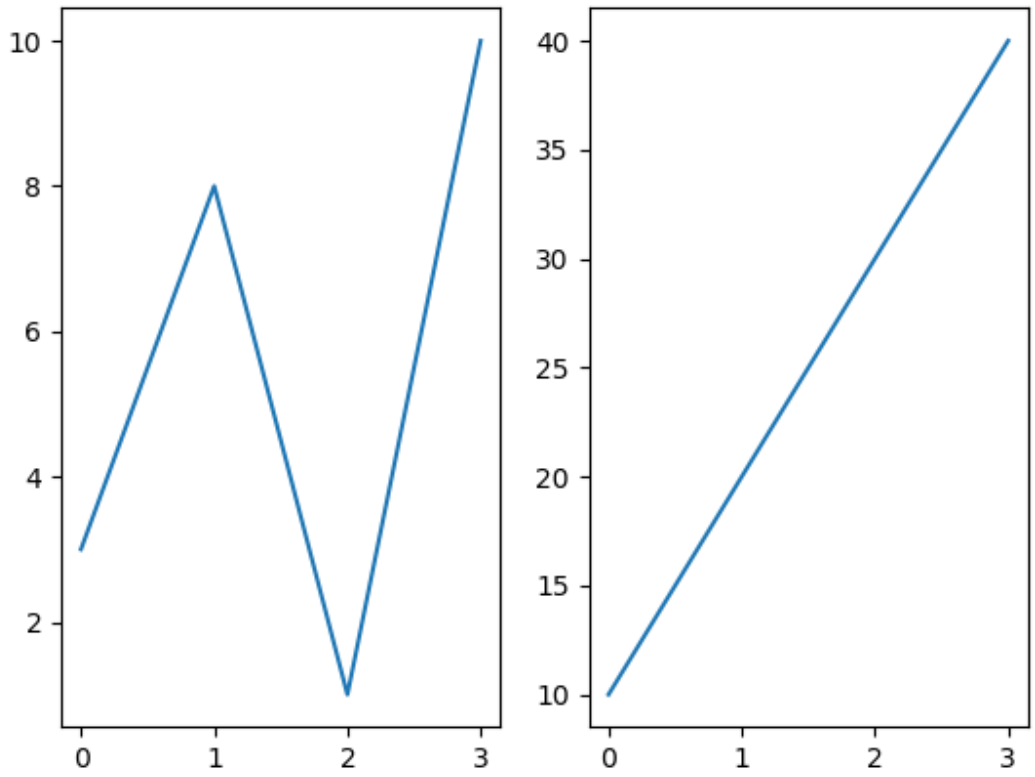



# Display Multiple Plots

With the `subplot()` function you can draw multiple plots in one 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)

#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()
```




# The subplot() Function

The `subplot()` 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.
```

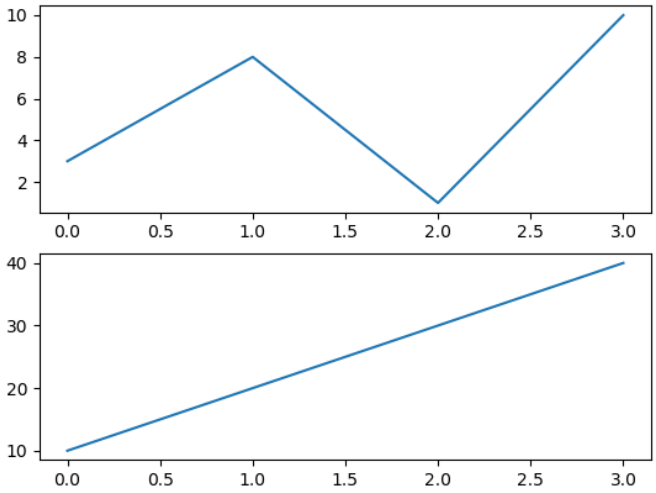
```
plt.subplot(1, 2, 2)
#the figure has 1 row, 2 columns, and this plot is the second plot.
```

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

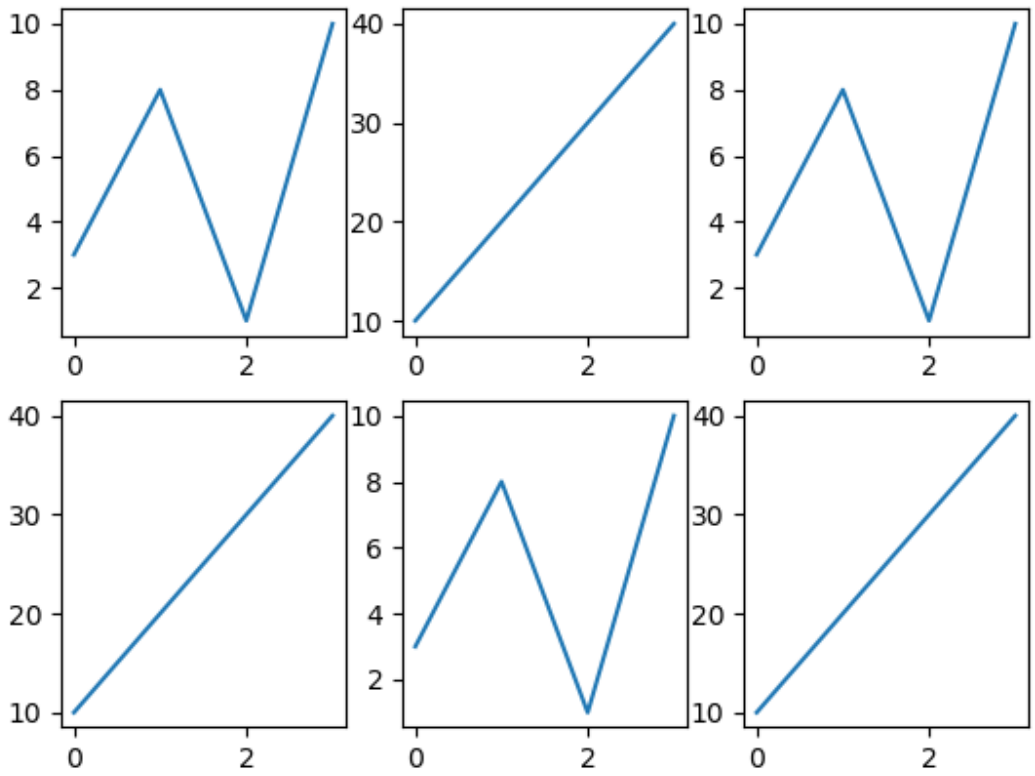
Draw 2 plots on top of each other:

```
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()
```



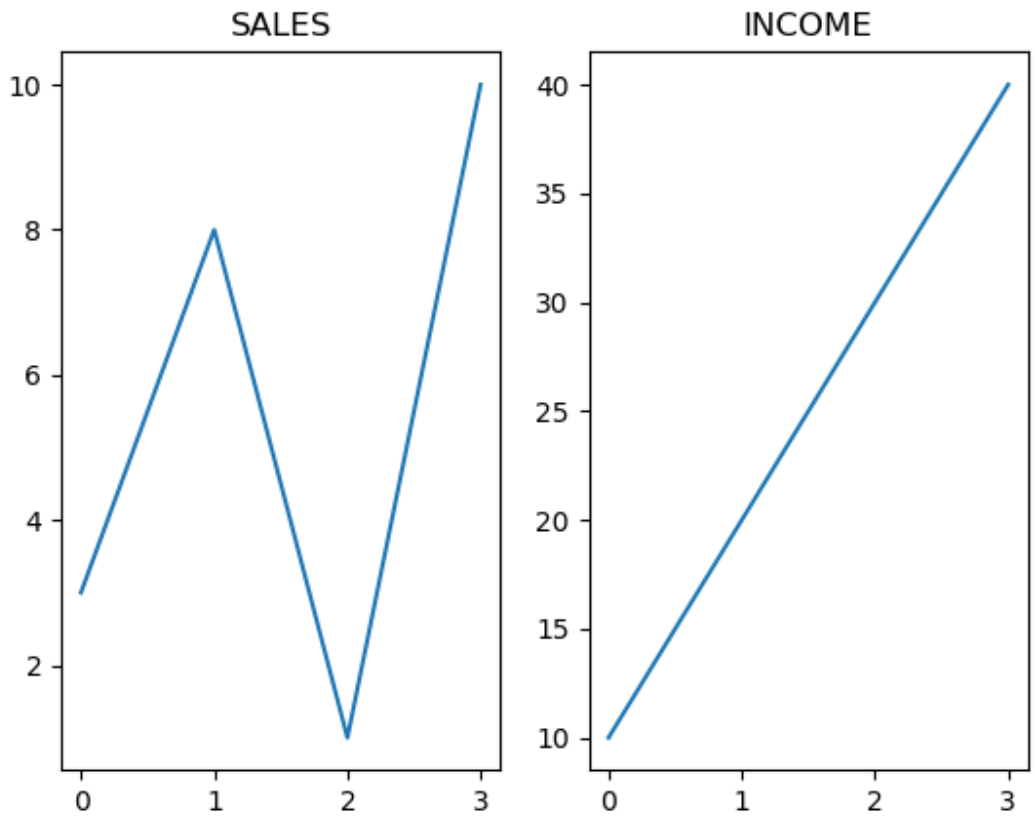
Draw 6 plots:

```
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()
```




```
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()
```




# Super Title

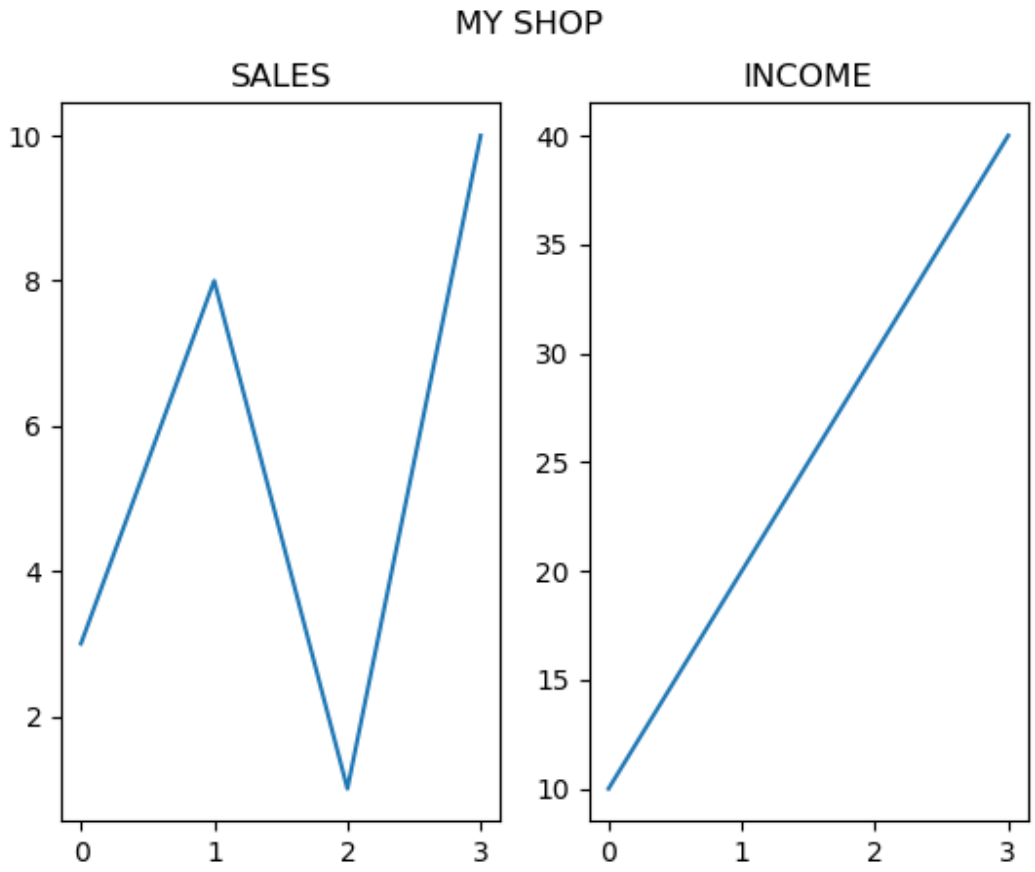
You can add a title to the entire figure with the `suptitle()` function:

## 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()
```



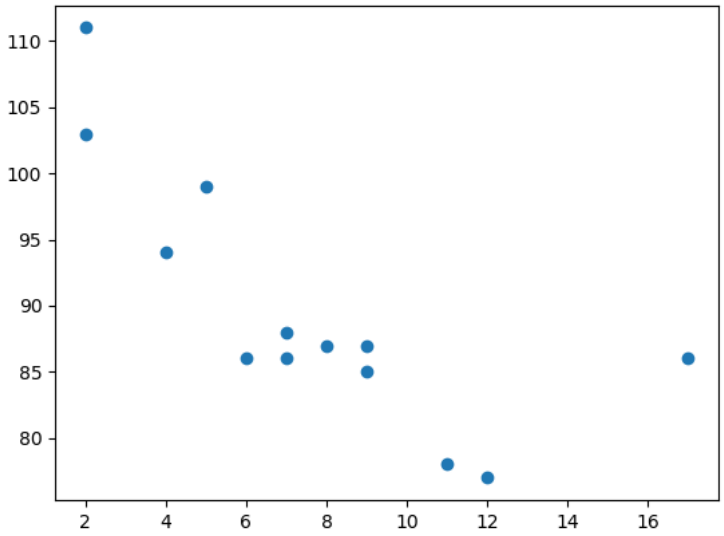

# Creating Scatter Plots

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:

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()
```




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.

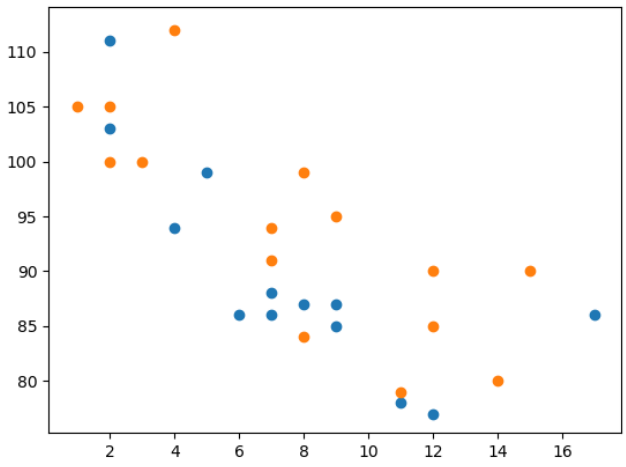
# Compare Plots

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()
```



**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.

---

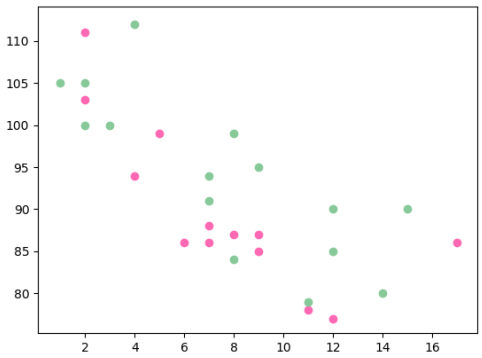
# Colors

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()
```



# Color Each Dot

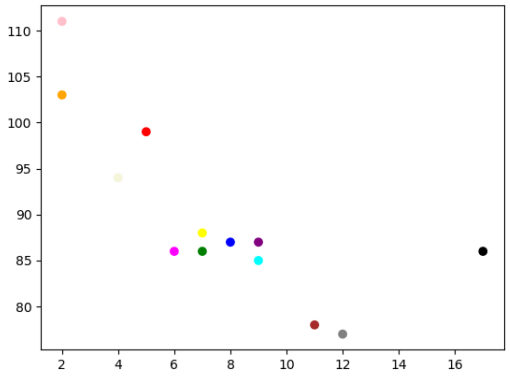
You can even set a specific color for each dot by using an array of colors as value for the `c` argument:

**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()
```




# 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, up to 100, which is a yellow color.

## How to Use the ColorMap

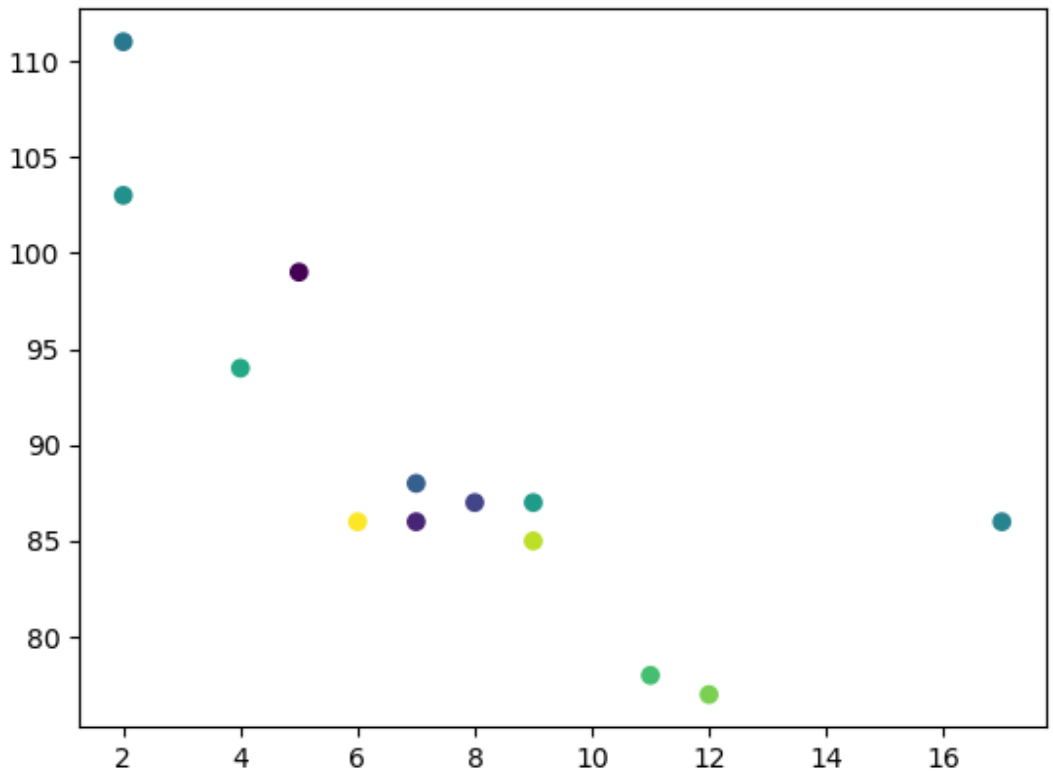
You can specify the colormap with the keyword argument `cmap` with the value of the colormap, in this case `'viridis'` which is one of the built-in colormaps available in Matplotlib.

In addition you have to create an array with values (from 0 to 100), one value for each 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()
```

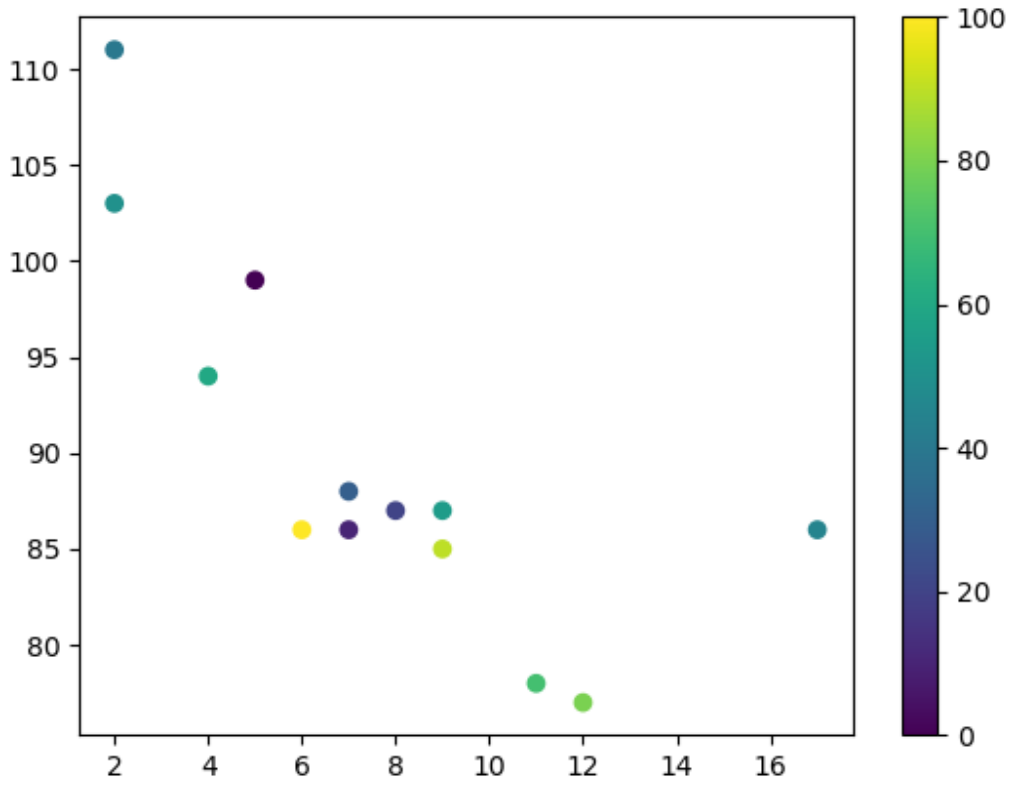



You can include the colormap in the drawing by including the `plt.colorbar()` statement:

## Example

Include the actual colormap:

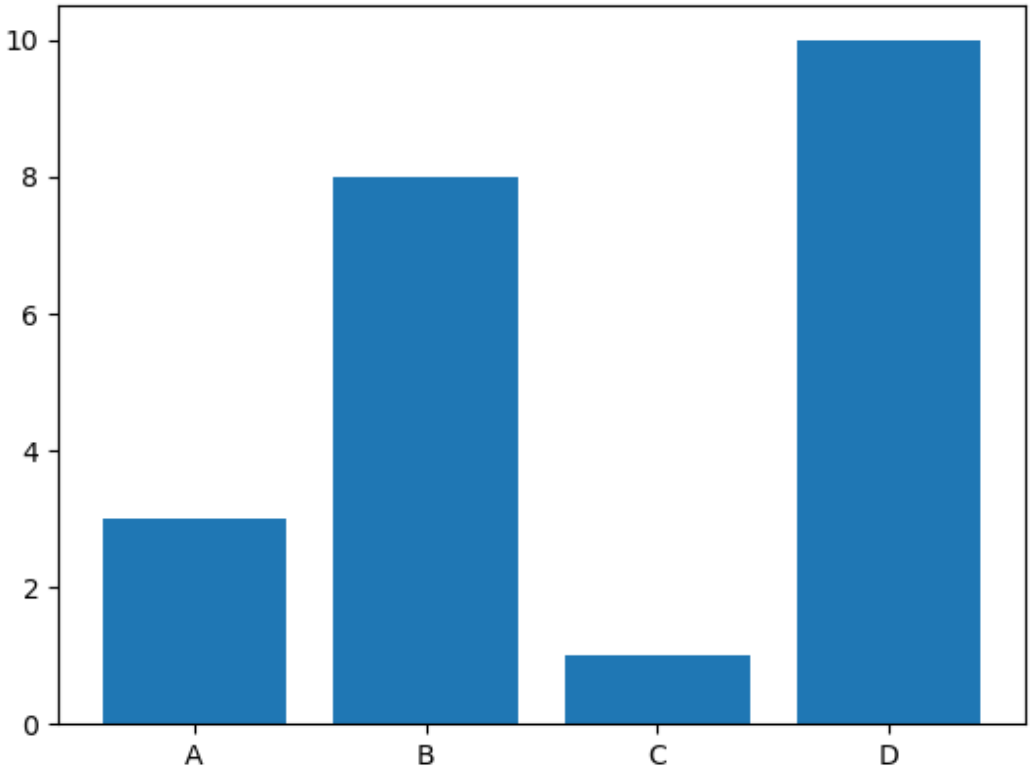
```
import matplotlib.pyplot as plt
import numpy as np

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()
```




# Creating Bars

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

```
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()
```




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)
```

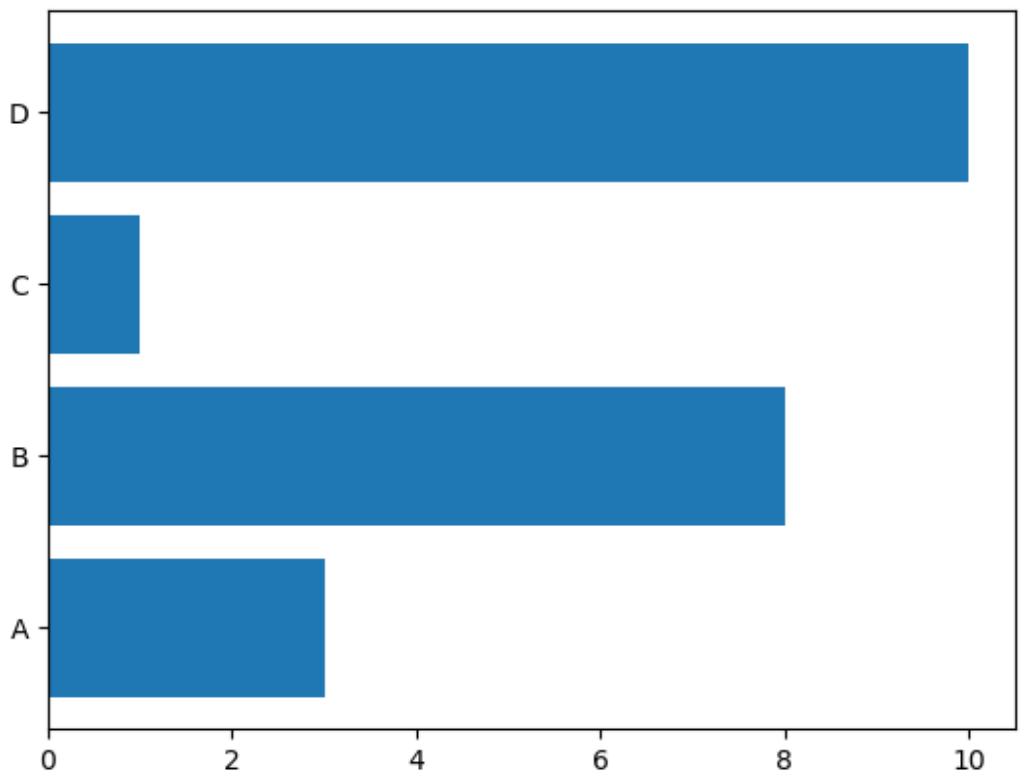
# Horizontal Bars

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()
```



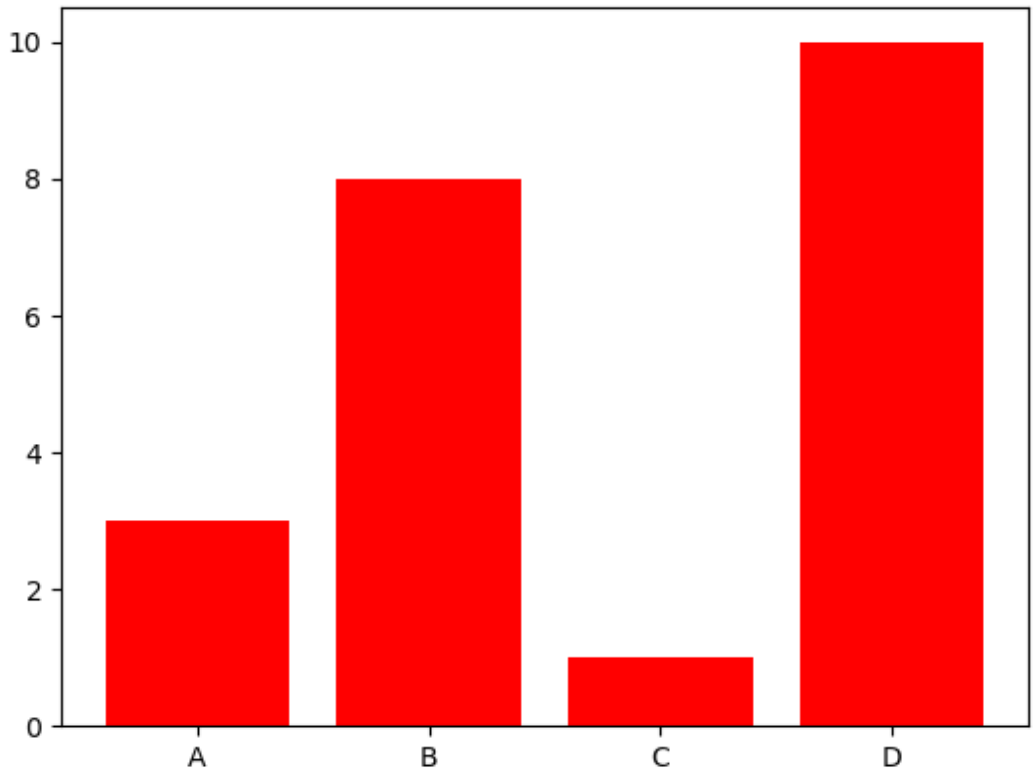
# ar Color

The `bar()` and `barh()` take 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()
```



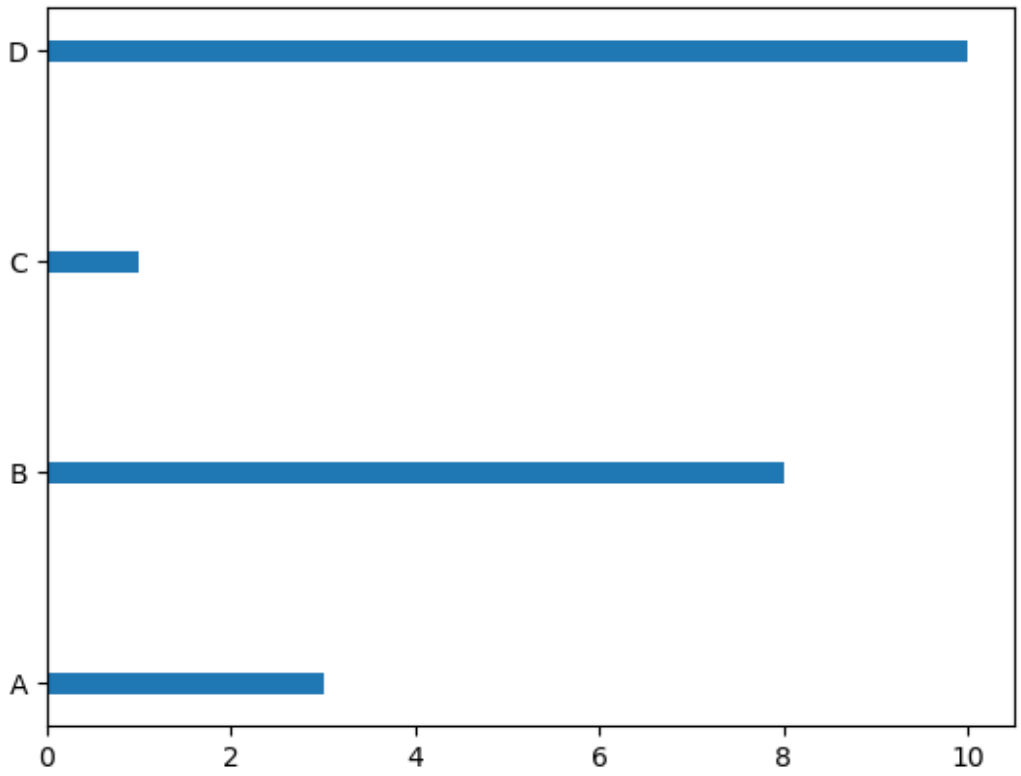
# Bar Height

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()
```

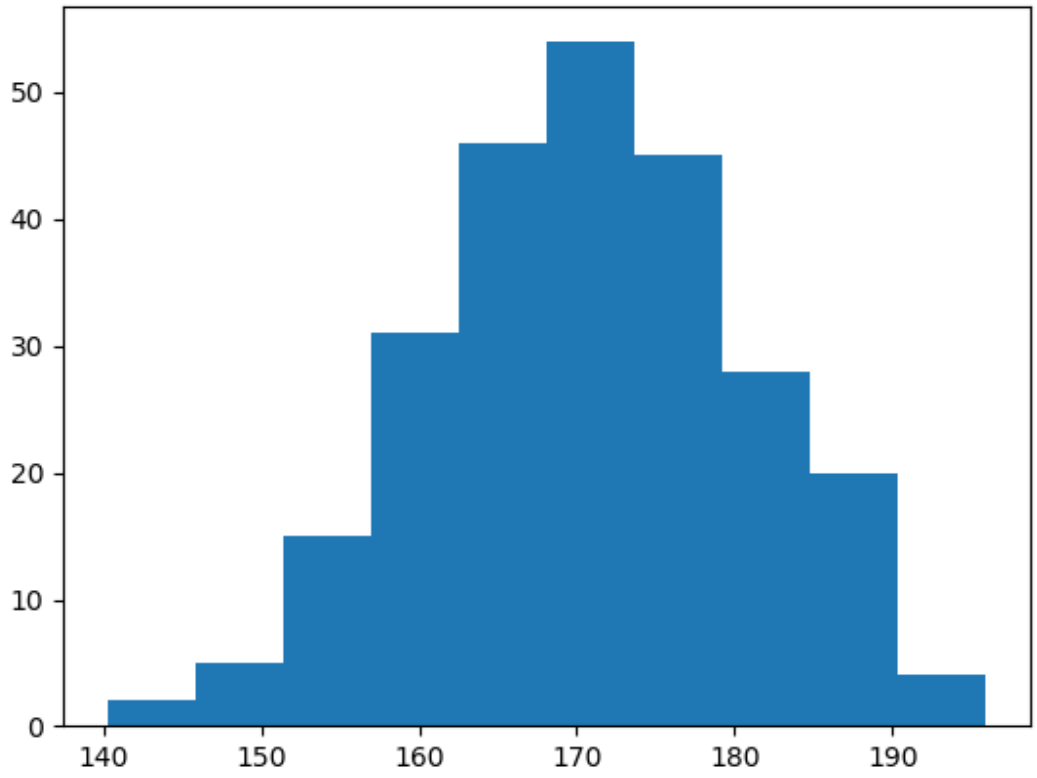


# Histogram

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.

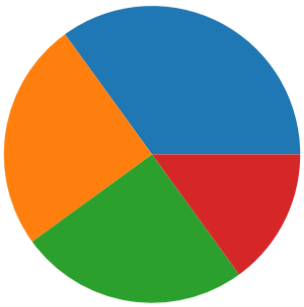
```
import numpy as np

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

print(x)
```

# Creating Pie Charts

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

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()
```



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 moves *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 `labels` parameter.

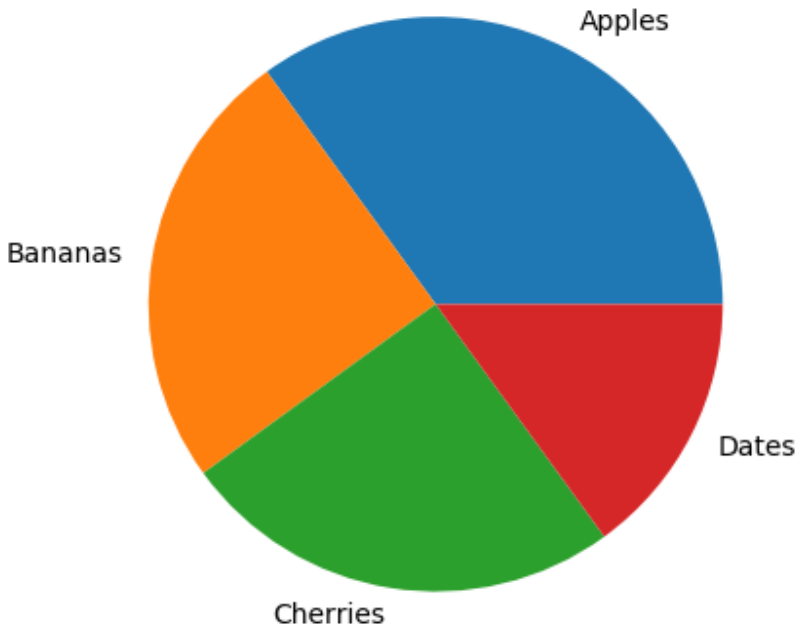
The `labels` parameter must be an array with one label for each wedge:

## Example

A simple pie chart:

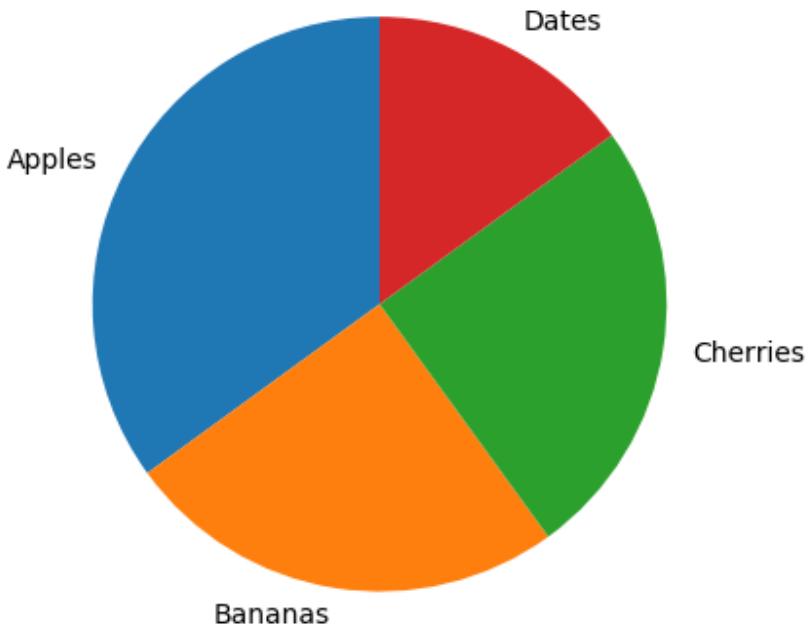
```
import matplotlib.pyplot as plt
import numpy as np
```

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

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




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()
```




# Explode

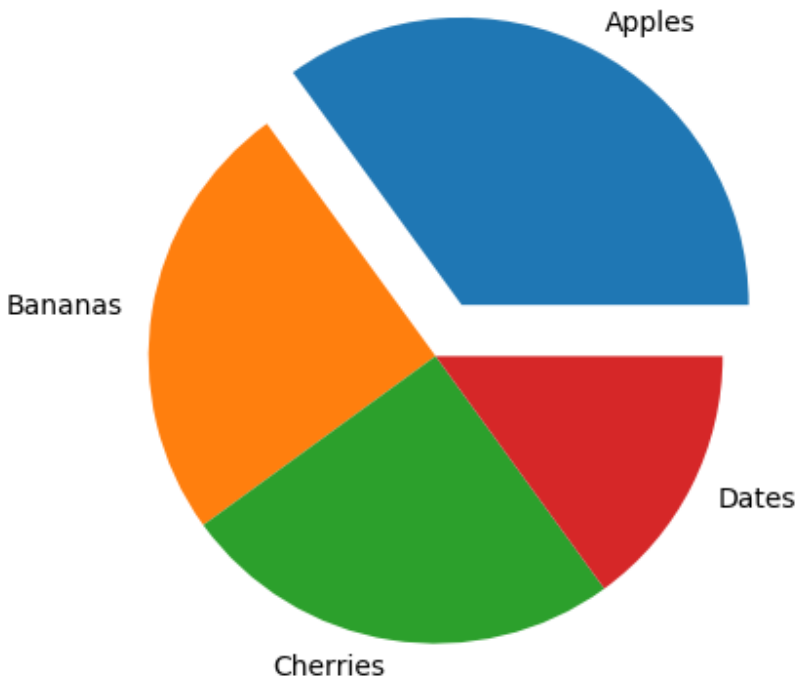
Maybe you want one of the wedges to stand out? The `explode` parameter allows you to do that.

The `explode` parameter, if specified, and not `None`, must be an array with one value for each wedge.

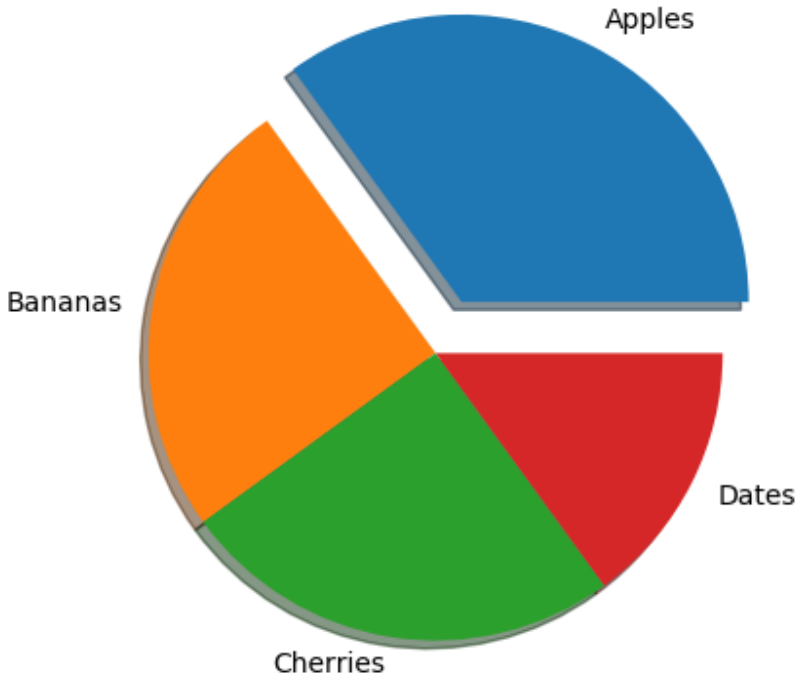
Each value represents how far from the center each wedge is displayed:

```
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()
```




```
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()
```




# Colors

You can set the color of each wedge with the `colors` parameter.

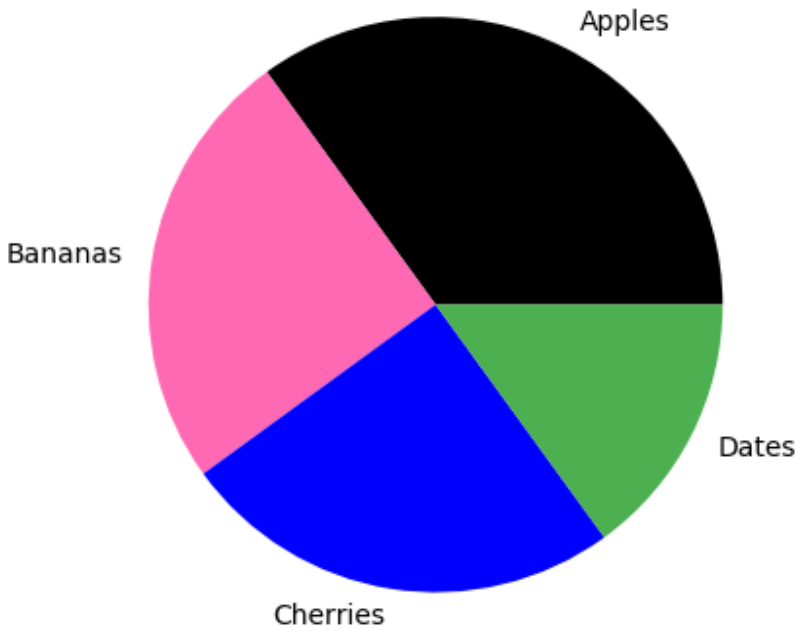
The `colors` parameter, if specified, must be an array with one value for each wedge:

## 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()
```




# Legend

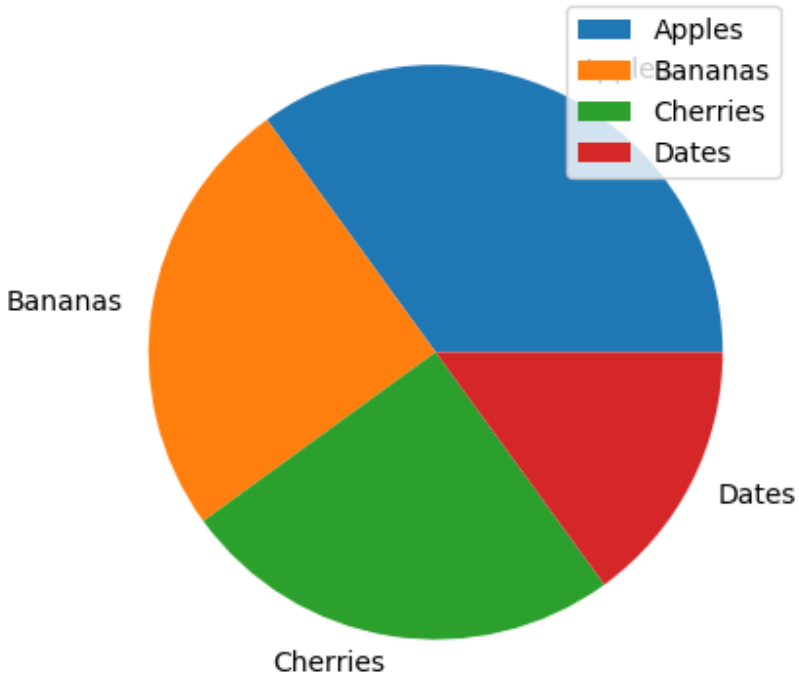
To add a list of explanation for each wedge, use the `legend()` function:

## 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()
```




## 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()
```