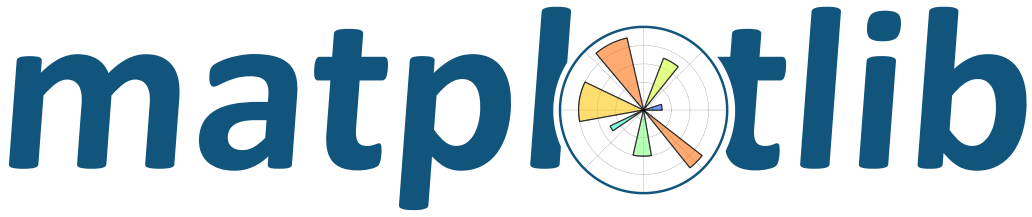


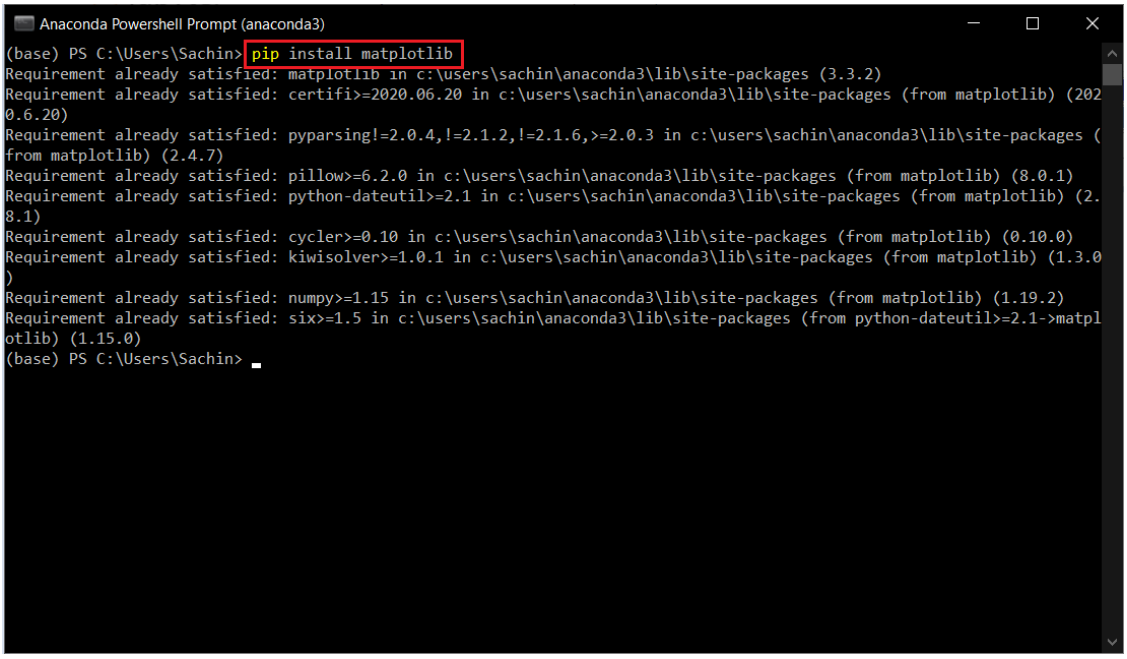
Matplotlib

Matplotlib is an amazing visualization library in Python for 2D plots of arrays. Matplotlib is a multi-platform data visualization library built on NumPy arrays and designed to work with the broader SciPy stack. It was introduced by John Hunter in the year 2002.

One of the greatest benefits of visualization is that it allows us visual access to huge amounts of data in easily digestible visuals. Matplotlib consists of several plots like line, bar, scatter, histogram etc.

Installing Matplotlib

To install Python Matplotlib, go to your command prompt and type "pip install matplotlib". Once the installation is completed, go to your IDE (For example: PyCharm) or Anaconda Jupyter Notebook and simply import it by typing: **import matplotlib as plt**.



How to import Matplotlib

In order to start using Matplotlib and all of the plot available in Matplotlib, You will need to import it. This can be easily done with this import statement.

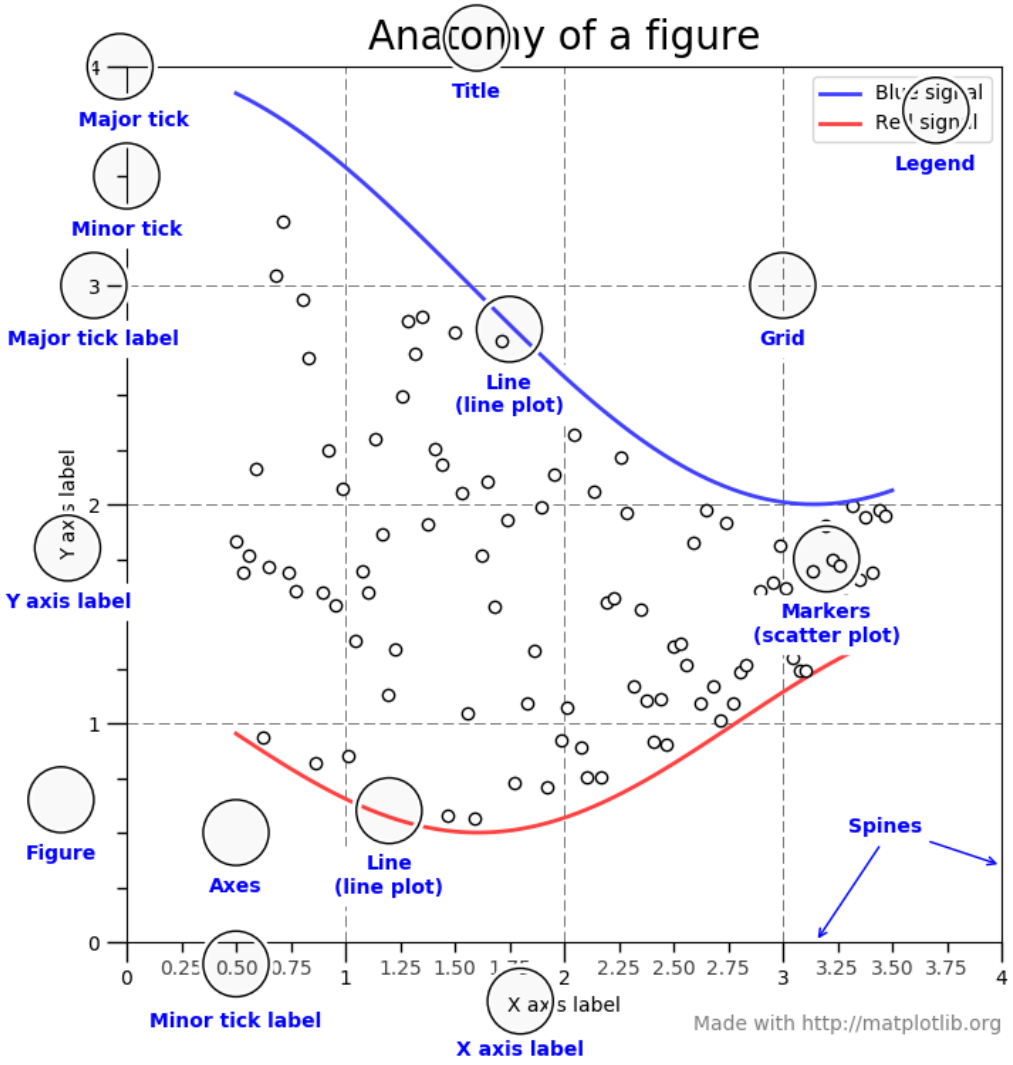
In [1]:
```
import matplotlib.pyplot as plt
```

NOTE: We shorten matplotlib to plt in order to save time and also to keep code

standardized so that anyone working with your code can easily understand and run it.

Parts of a figure

First Let's have a look what is the attribute we have in the graph. Then we dig into what is matplotlib and how to use it.



Legends:

In [ ]:

```
Location string    Location code

Best                0
upper right         1
upper left          2
lower left          3
lower right         4
Right               5
Center left         6
Center right        7
lower center        8
upper center        9
Center              10
```

Color codes:

In [ ]:
```
Character       Color

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

Makers code:

In [ ]:
```
Character       Description

'.'             point marker
','             pixel marker
'o'             circle marker
'v'             triangle_down marker
'^'             triangle_up marker
'<'             triangle_left marker
'>'             triangle_right marker
'1'             tri_down marker
'2'             tri_up marker
'3'             tri_left marker
'4'             tri_right marker
'8'             octagon marker
's'             square marker
'p'             pentagon marker
'P'             plus (filled) marker
'*'             star marker
'h'             hexagon1 marker
'H'             hexagon2 marker
'+'             plus marker
'x'             x marker
'X'             x (filled) marker
'D'             diamond marker
'd'             thin_diamond marker
'|'             vline marker
'_'             hline marker
```

Line Styles:

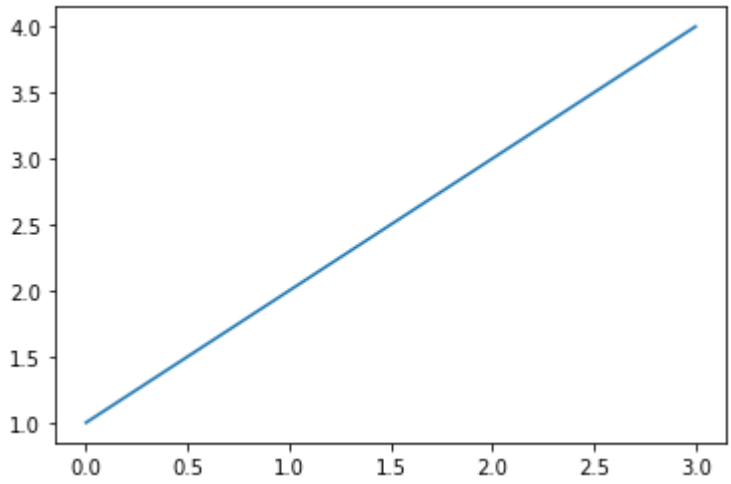
In [ ]:
```
Character       Description

'-'             Solid line
'--'            Dashed line
'-.'            Dash-dot line
':'             Dotted line
'H'             Hexagon marker
```

Line plot

A line graph is a unique graph which is commonly used in matplotlib. It represents the change in a quantity with respect to another quantity. Line graph is usually plotted in a two-dimensional XY plane. If the relation including any two measures can be expressed utilizing a straight line in a graph, then such graphs are called linear graphs. Thus, the line graph is also called a linear graph.

In [2]:
```
plt.plot([1, 2, 3, 4])
plt.show()
```

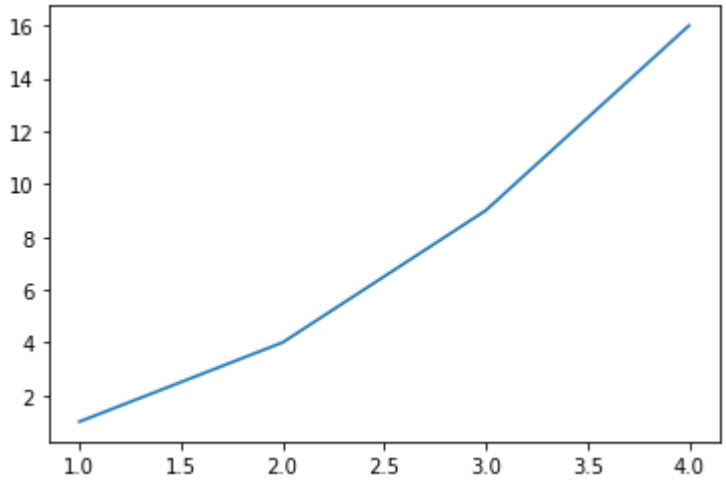


You may be wondering why the x-axis ranges from 0-3 and the y-axis from 1-4. If you provide a single list or array to plot, matplotlib assumes it is a sequence of y values, and automatically generates the x values for you. Since python ranges start with 0, the default x vector has the same length as y but starts with 0. Hence the x data are [0, 1, 2, 3].

plot is a versatile function, and will take an arbitrary number of arguments. For example, to plot x versus y, you can write:

In [3]:
```
plt.plot([1, 2, 3, 4], [1, 4, 9, 16])
```

Out[3]:
```
[<matplotlib.lines.Line2D at 0x1fa38113130>]
```

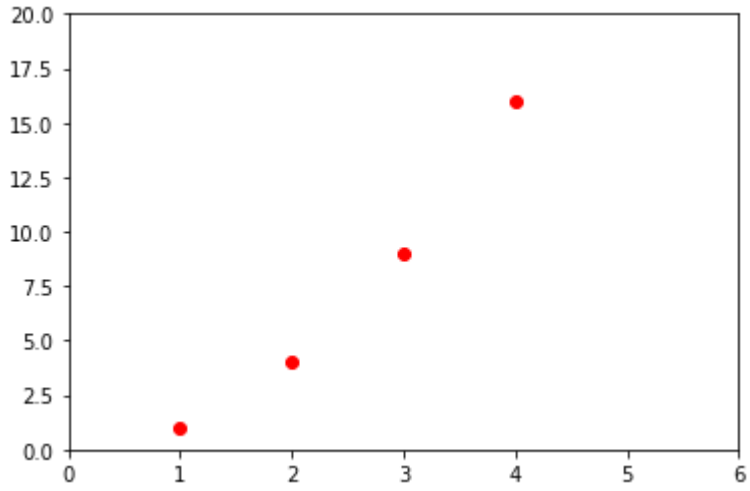


Formatting the style of your plot

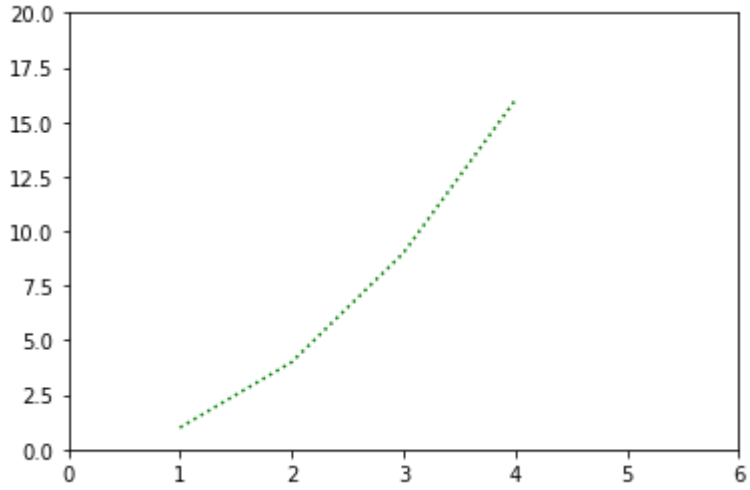
For every x, y pair of arguments, there is an optional third argument which is the format string that indicates the color and line type of the plot. The letters and symbols of the format string are from MATLAB, and you concatenate a color string with a line style string.

The default format string is 'b-', which is a solid blue line. For example, to plot the above with red circles.

In [4]:
```
plt.plot([1, 2, 3, 4], [1, 4, 9, 16], 'ro')
plt.axis([0, 6, 0, 20])
plt.show()
```



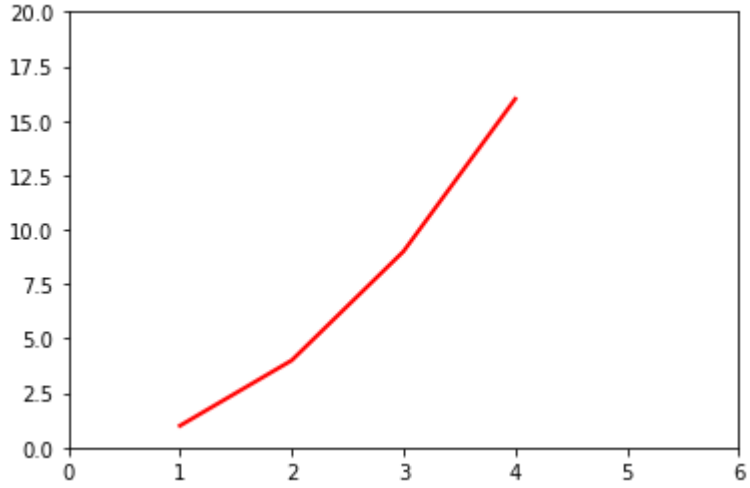
In [5]:
```
plt.plot([1, 2, 3, 4], [1, 4, 9, 16], 'g:')
plt.axis([0, 6, 0, 20])
plt.show()
```



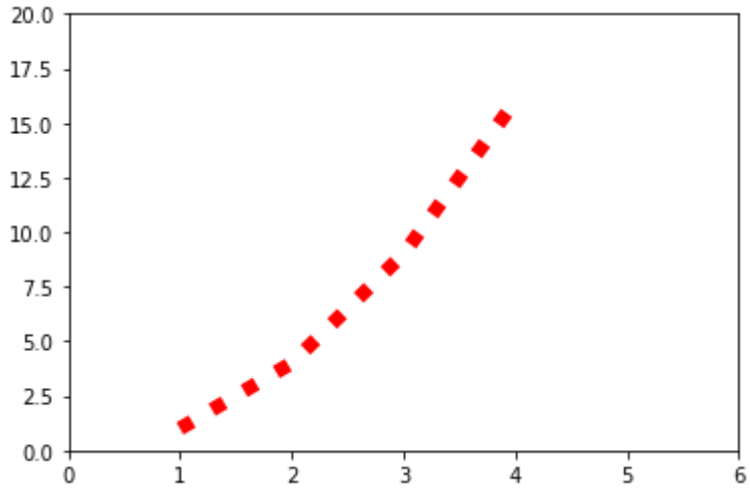
Controlling line properties

Lines have many attributes that you can set: linewidth, dash style etc. There are several ways to set line properties

In [6]:
```
plt.plot([1, 2, 3, 4], [1, 4, 9, 16], 'r-', linewidth=2.0)
plt.axis([0, 6, 0, 20])
plt.show()
```



In [7]:
```
plt.plot([1, 2, 3, 4], [1, 4, 9, 16], 'r:', linewidth=7.0)
plt.axis([0, 6, 0, 20])
plt.show()
```

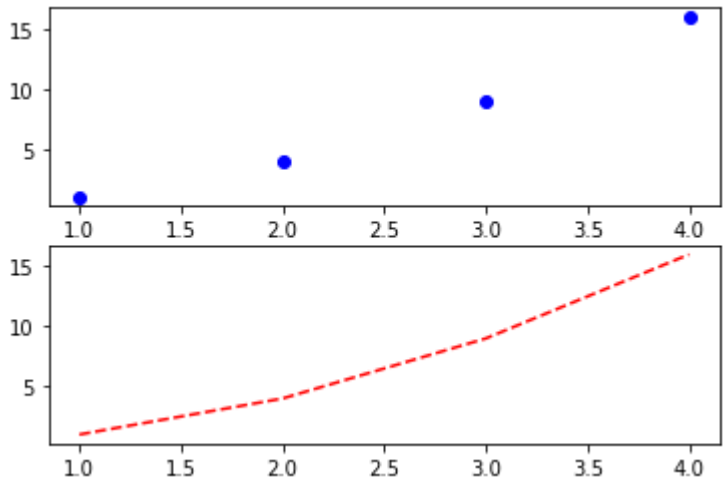


Working with multiple figures and axes

pyplot, have the concept of the current figure and the current axes. All plotting functions apply to the current axes. The function gca returns the current axes (a matplotlib.axes.Axes instance), and gcf returns the current figure (a matplotlib.figure.Figure instance). Normally, you don't have to worry about this, because it is all taken care of behind the scenes. Below is a script to create two subplots.

In [8]:
```
plt.figure()
plt.subplot(211)
plt.plot([1, 2, 3, 4], [1, 4, 9, 16], 'bo')

plt.subplot(212)
plt.plot([1, 2, 3, 4], [1, 4, 9, 16], 'r--')
plt.show()
```

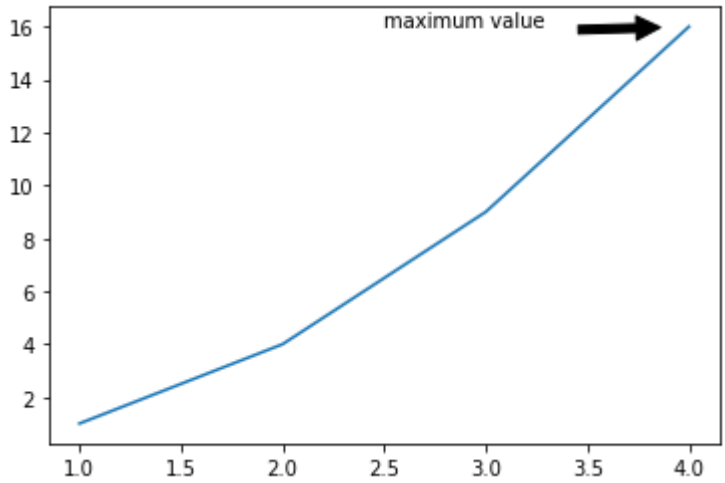


Annotating text

The uses of the basic text function above place text at an arbitrary position on the Axes. A common use for text is to annotate some feature of the plot, and the annotate method provides helper functionality to make annotations easy. In an annotation, there are two points to consider: the location being annotated represented by the argument xy and the location of the text xytext. Both of these arguments are (x, y) tuples.

In [9]:
```
plt.plot([1, 2, 3, 4], [1, 4, 9, 16])
plt.annotate('maximum value', xy=(4, 16), xytext=(2.5, 16),arrowprops=di
```

Out[9]:
```
Text(2.5, 16, 'maximum value')
```



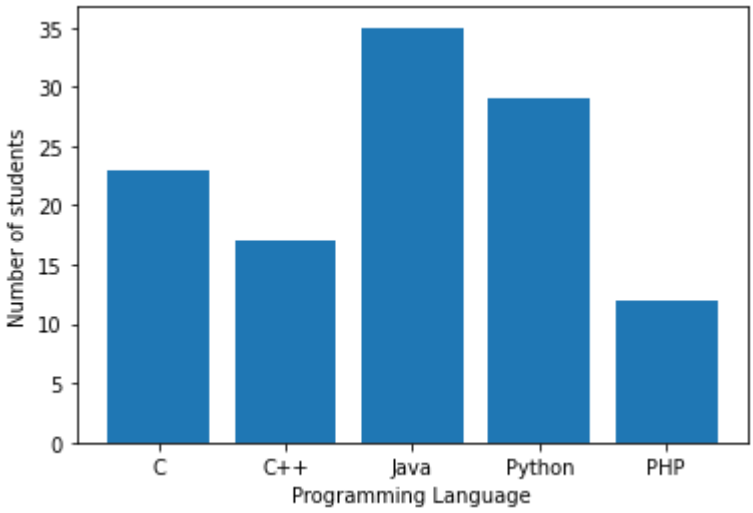
In this basic example, both the xy (arrow tip) and xytext locations (text location) are in data coordinates. There are a variety of other coordinate systems one can choose -- see Basic annotation and Advanced Annotations for details. More examples can be found in Annotating Plots.

Bar Graph

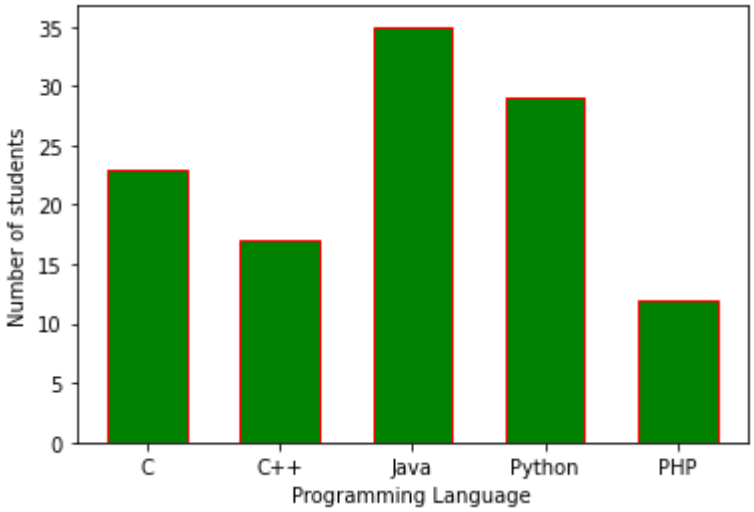
Bar graphs are the pictorial representation of data (generally grouped), in the form of vertical or horizontal rectangular bars, where the length of bars are proportional to the measure of data. They are also known as bar charts

In [10]:
```
langs = ['C', 'C++', 'Java', 'Python', 'PHP']
students = [23,17,35,29,12]
```

```
plt.bar(langs,students)
# naming the x axis
plt.xlabel("Programming Language")
# naming the y axis
plt.ylabel("Number of students")
plt.show()
```



In [11]:
```
langs = ['C', 'C++', 'Java', 'Python', 'PHP']
students = [23,17,35,29,12]
plt.bar(langs,students,width=0.6,color="green",edgecolor="red")
# naming the x axis
plt.xlabel("Programming Language")
# naming the y axis
plt.ylabel("Number of students")
plt.show()
```

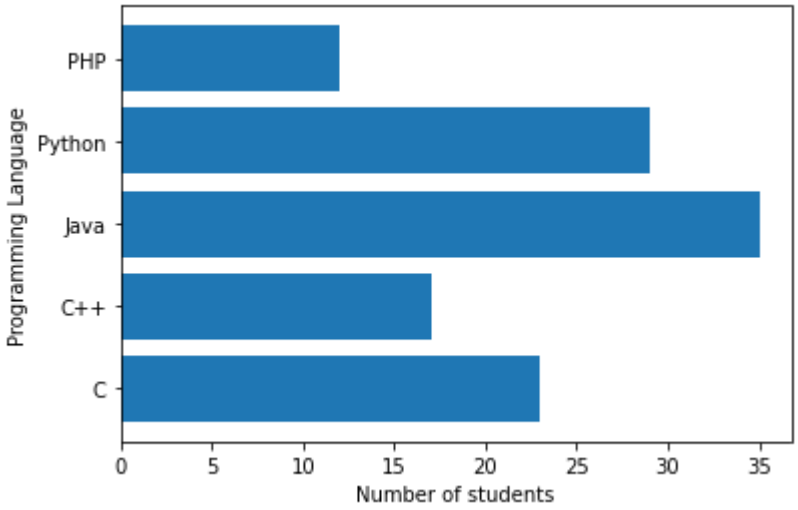


Horizontal Bar Graph

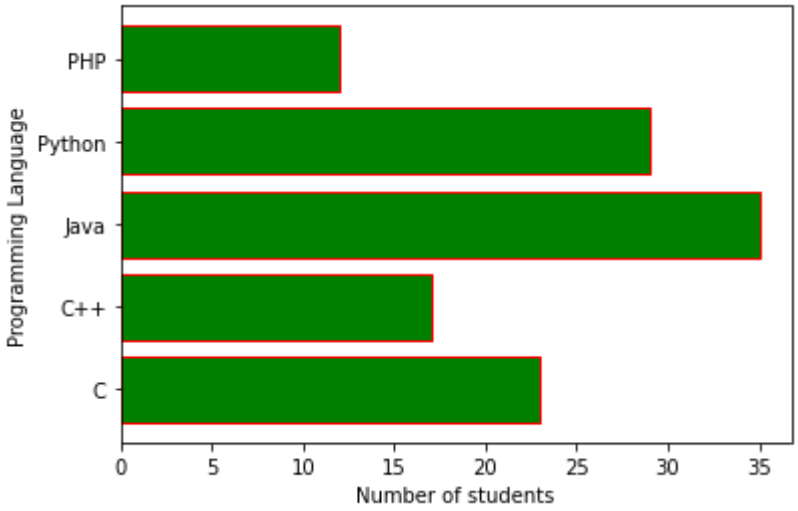
When the grouped data are represented horizontally in a chart with the help of bars, then such graphs are called horizontal bar graphs, where the bars show the measure of data. The data is depicted here along the x-axis of the graph, and the length of the bars denote the values.

In [12]:
```
langs = ['C', 'C++', 'Java', 'Python', 'PHP']
students = [23,17,35,29,12]
plt.barh(langs,students)
```

```
# naming the y axis
plt.xlabel("Number of students")
# naming the x axis
plt.ylabel("Programming Language")
plt.show()
```



In [13]:
```
langs = ['C', 'C++', 'Java', 'Python', 'PHP']
students = [23,17,35,29,12]
plt.barh(langs,students,color="green",edgecolor="red")
# naming the y axis
plt.xlabel("Number of students")
# naming the x axis
plt.ylabel("Programming Language")
plt.show()
```



When to use Bar Graph

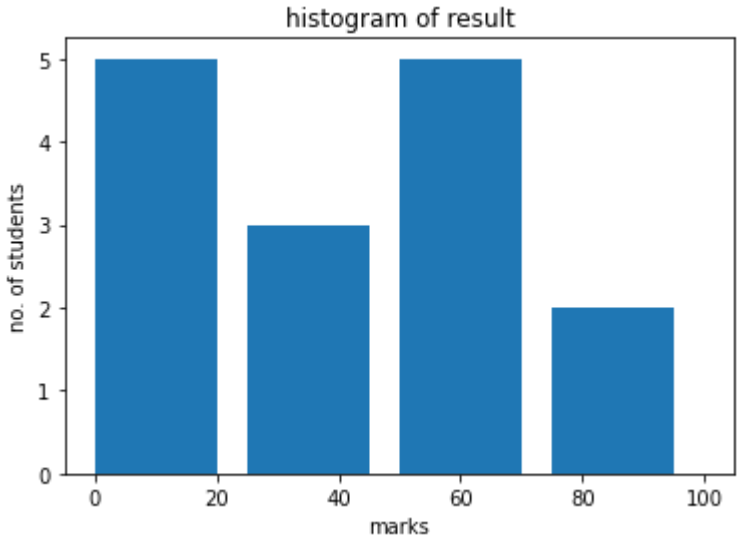
The bar graph is used to compare the items between different groups over time. Bar graphs are used to measure the changes over a period of time. When the changes are larger, a bar graph is the best option to represent the data.

Histogram

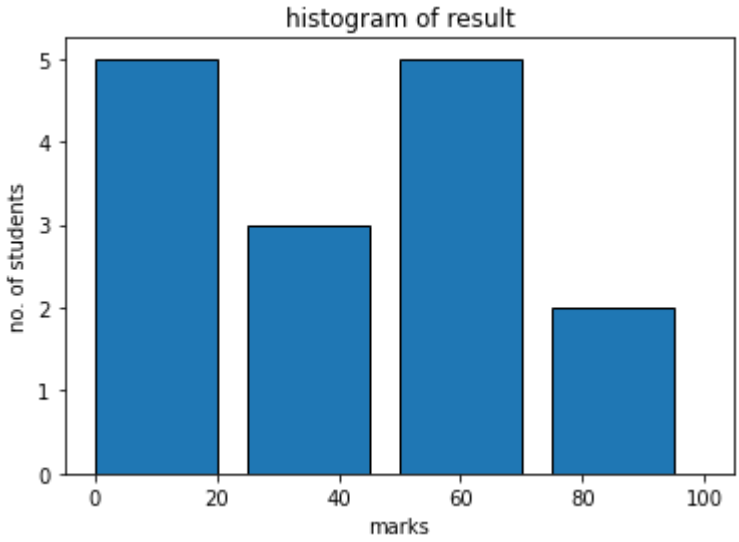
When the grouped data are represented horizontally in a chart with the help of bars, then such graphs are called horizontal bar graphs, where the bars show the measure of data.

The data is depicted here along the x-axis of the graph, and the length of the bars denote the values.

In [15]:
```
import numpy as np
a = np.array([22,87,5,43,56,73,55,54,11,20,51,5,79,31,27])
plt.hist(a, bins = [0,25,50,75,100],width=20,)
plt.title("histogram of result")
plt.xlabel('marks')
plt.ylabel('no. of students')
plt.show()
```



In [16]:
```
a = np.array([22,87,5,43,56,73,55,54,11,20,51,5,79,31,27])
plt.hist(a, bins = [0,25,50,75,100],width=20,edgecolor="black")
plt.title("histogram of result")
plt.xlabel('marks')
plt.ylabel('no. of students')
plt.show()
```



When to use histogram

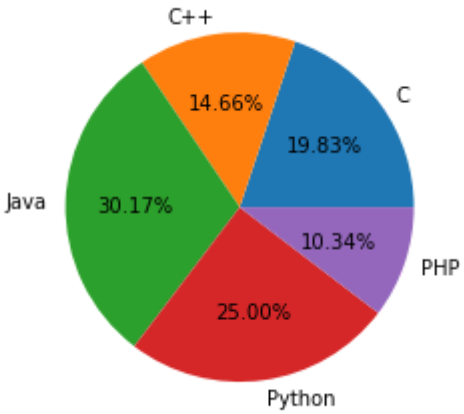
- The data should be numerical.
- A histogram is used to check the shape of the data distribution.
- Used to check whether the process changes from one period to another.

- Used to determine whether the output is different when it involves two or more processes.

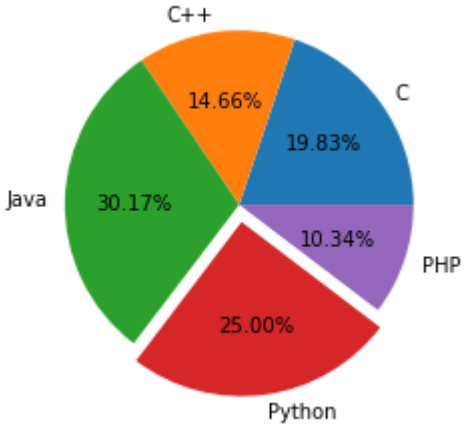
Piechart

The “pie chart” is also known as “circle chart”, that divides the circular statistical graphic into sectors or slices in order to illustrate the numerical problems. Each sector denotes a proportionate part of the whole. To find out the composition of something, Pie-chart works the best at that time. In most cases, pie charts replace some other graphs like the bar graph, line plots, histograms, etc.

In [17]:
```
plt.axis('equal')
langs = ['C', 'C++', 'Java', 'Python', 'PHP']
students = [23,17,35,29,12]
plt.pie(students, labels = langs,autopct='%1.2f%%')
# function to show the plot
plt.show()
```



In [18]:
```
langs = ['C', 'C++', 'Java', 'Python', 'PHP']
students = [23,17,35,29,12]
explode=(0,0,0,0.1,0)
plt.pie(students, explode=explode, labels = langs,autopct='%1.2f%%')
# function to show the plot
plt.show()
```



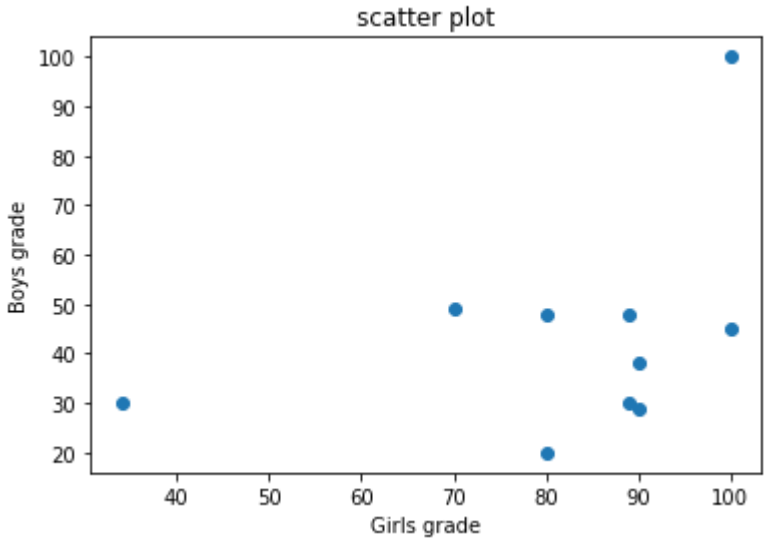
When to use piechart

It is used to compare to represent categorical data.

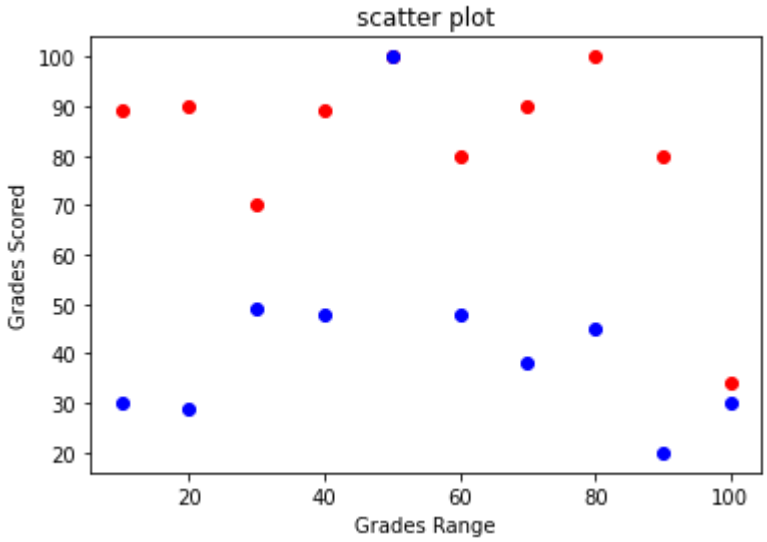
Scatter Plot

Scatter plots are the graphs that present the relationship between two variables in a data-set. It represents data points on a two-dimensional plane or on a Cartesian system. The independent variable or attribute is plotted on the X-axis, while the dependent variable is plotted on the Y-axis. These plots are often called scatter graphs or scatter diagrams.

In [19]:
```
girls_grades = [89, 90, 70, 89, 100, 80, 90, 100, 80, 34]
boys_grades = [30, 29, 49, 48, 100, 48, 38, 45, 20, 30]
plt.scatter(girls_grades, boys_grades)
plt.xlabel("Girls grade")
plt.ylabel("Boys grade")
plt.title('scatter plot')
plt.show()
```



In [20]:
```
girls_grades = [89, 90, 70, 89, 100, 80, 90, 100, 80, 34]
boys_grades = [30, 29, 49, 48, 100, 48, 38, 45, 20, 30]
grades_range = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
plt.scatter(grades_range, girls_grades, color='r')
plt.scatter(grades_range, boys_grades, color='b')
plt.xlabel('Grades Range')
plt.ylabel('Grades Scored')
plt.title('scatter plot')
plt.show()
```



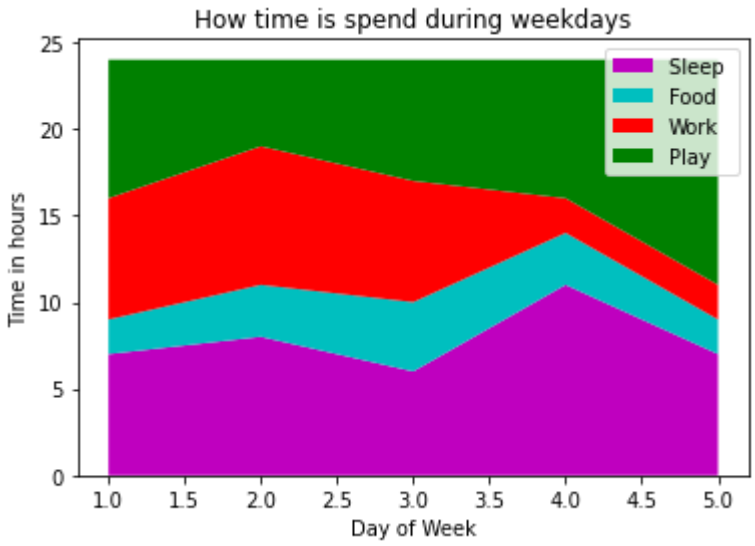
When to use Scatter plot

When the data is in numeric form we use the scatter plot to compare pair of values.

Stack plot

Stackplot is used to draw a stacked area plot. It displays the complete data for visualization. It shows each part stacked onto one another and how each part makes the complete figure. It displays various constituents of data and it behaves like a pie chart.

In [21]:
```
import numpy as np
days=[1,2,3,4,5]
sleep=[7,8,6,11,7]
food=[2,3,4,3,2]
work=[7,8,7,2,2]
play=[8,5,7,8,13]
lab = ["Sleep ", "Food", "Work","Play"]
fig, ax = plt.subplots()
ax.stackplot(days, sleep, food, work,play, labels=lab,colors=['m','c','r
ax.legend(loc='upper right')
# naming the x axis
plt.xlabel('Day of Week')
# naming the y axis
plt.ylabel('Time in hours')
# giving a title to my graph
plt.title('How time is spend during weekdays')
# function to show the plot
plt.show()
```

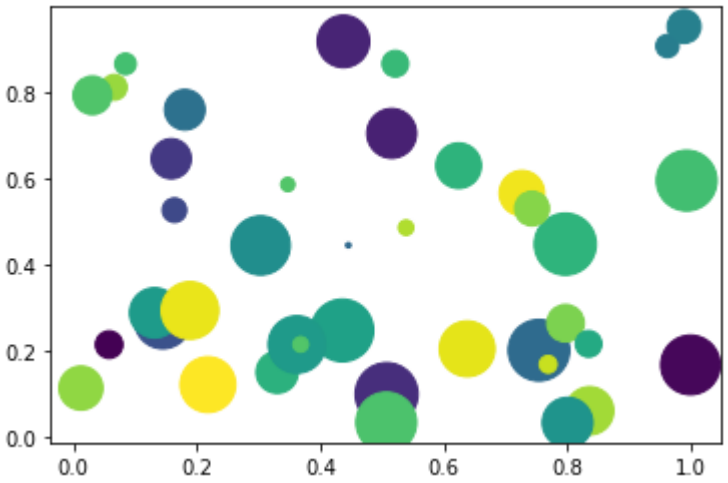


Bubble chart

A bubble chart is a data visualization that displays multiple circles (bubbles) in a two-dimensional plot. It is a generalization of the scatter plot, replacing the dots with bubbles. Most commonly, a bubble chart displays the values of three numeric variables, where each observation's data is shown by a circle ("bubble"), while the horizontal and vertical positions of the bubble show the values of two other variables.

In [22]:
```
x = np.random.rand(40)
y = np.random.rand(40)
z = np.random.rand(40)
```

```
colors = np.random.rand(40)
plt.scatter(x, y, s=z*1000,c=colors)
plt.show()
```



Explore via coding

In [23]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

iris=pd.read_csv("D:\Python learning track and notes\Module 1\Data Analy
```

In [24]:
```
iris
```

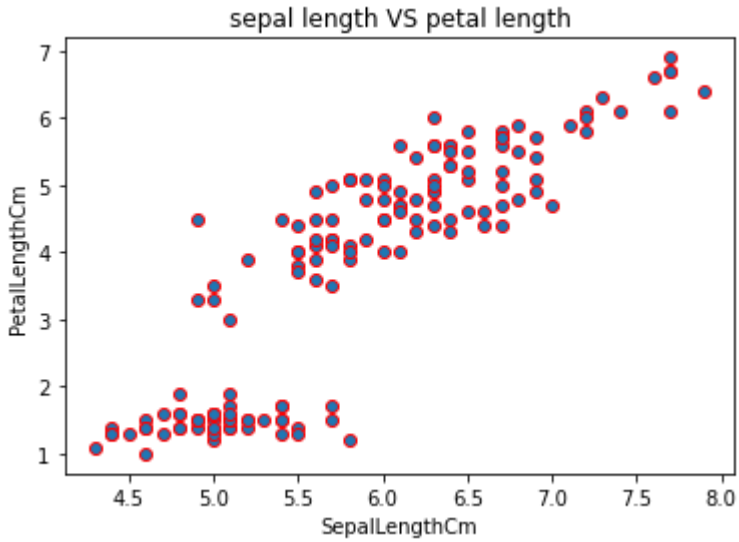
Out[24]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...	...
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 6 columns

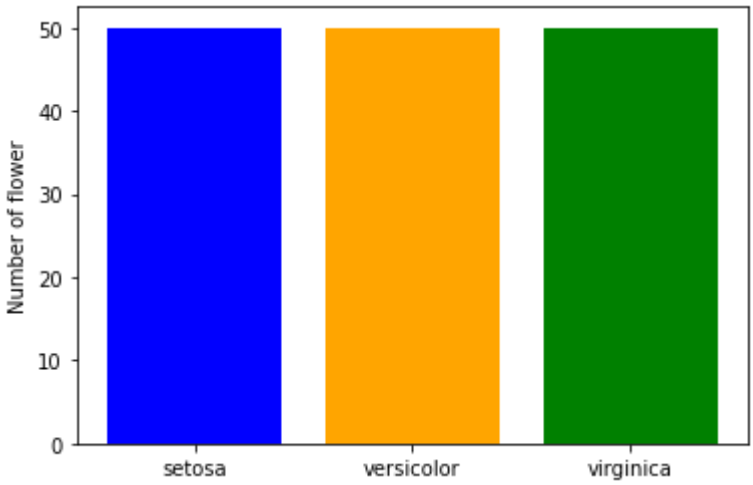
Create a scatter plot between features sepal length and petal length

In [25]:
```
plt.scatter(x=iris['SepalLengthCm'],y=iris['PetalLengthCm'],edgecolors="
plt.xlabel("SepalLengthCm")
plt.ylabel("PetalLengthCm")
plt.title("sepal length VS petal length")
plt.show()
```



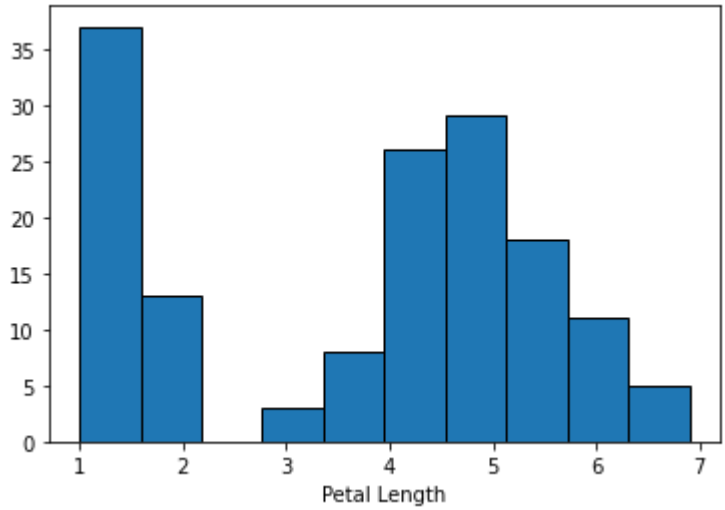
Create a bar graph that shows how many types of flower present in data.

In [26]:
```
iris_count_versicolor,iris_count_virginica,iris_count_setosa=iris['Speci
plt.bar("setosa",iris_count_setosa,color="blue")
plt.bar("versicolor",iris_count_versicolor,color="orange")
plt.bar("virginica",iris_count_virginica,color="green")
plt.ylabel("Number of flower")
plt.show()
```



Create a histogram of petal length.

In [27]:
```
plt.hist(iris['PetalLengthCm'],edgecolor="black")
plt.xlabel("Petal Length")
plt.show()
```



Create a piechart to represent to show how many data point we have to each flower type.

In [28]:
```
iris_count_versicolor,iris_count_virginica,iris_count_setosa=iris['Speci
count=[iris_count_setosa,iris_count_versicolor,iris_count_virginica]
flower_type=['Setosa','Versicolor','Virginica']
plt.pie(count,labels=flower_type,autopct="%1.2f%%")
```

Out[28]:
```
([<matplotlib.patches.Wedge at 0x1fa3a0578e0>,
  <matplotlib.patches.Wedge at 0x1fa3a057f70>,
  <matplotlib.patches.Wedge at 0x1fa3a066640>],
 [Text(0.5499999702695115, 0.9526279613277875, 'Setosa'),
  Text(-1.0999999999999954, -1.0298943258065002e-07, 'Versicolor'),
  Text(0.5500001486524352, -0.9526278583383436, 'Virginica')],
 [Text(0.2999999837833699, 0.5196152516333385, '33.33%'),
  Text(-0.5999999999999974, -5.6176054134900006e-08, '33.33%'),
  Text(0.30000008108314646, -0.5196151954572783, '33.33%')])
```