

TOPIC-6: Visualizing Data with Graphs- Plotting with matplotlib-2D plots- plotting the scalar and vector fields, Scatter Plots, and Graph customization.

2-D Plot

'Matplotlib' is a Python library for publication-quality 2D and 3D graphics, with support for a variety of different output formats. More information about Matplotlib is available at the project's web site www.matplotlib.org. This web site contains detailed documentation and an extensive gallery that showcases the various types of graphs that can be generated using the Matplotlib library, together with the code for each example.

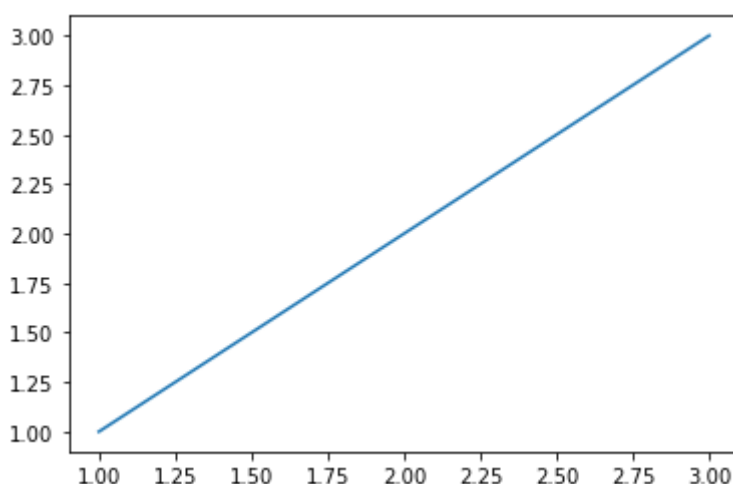
Matplotlib is imported using the following standard convention:

```
In [16]: import matplotlib.pyplot as plt
```

The import statement 'import matplotlib.pyplot as plt', is for convenient access to the submodule 'matplotlib.pyplot' that provides the functions that we will use to create new Figure instances.

Plot the points (1,1), (2,2), (3,3).

```
In [17]: x = [1, 2, 3]
y = [1, 2, 3]
plt.plot(x, y)
plt.show()
```



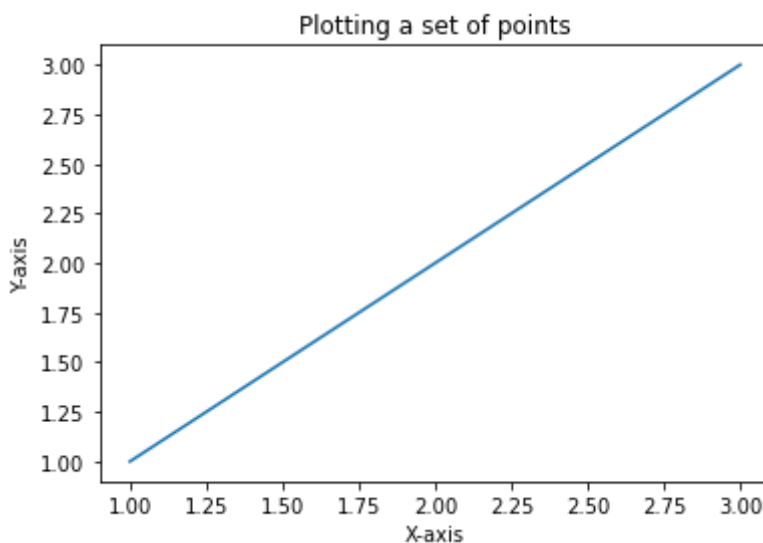
Adding the axis-labels, figure-title

```
In [5]: x = [1, 2, 3]
y = [1, 2, 3]

plt.plot(x, y)
```

```
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Plotting a set of points')

plt.show()
```



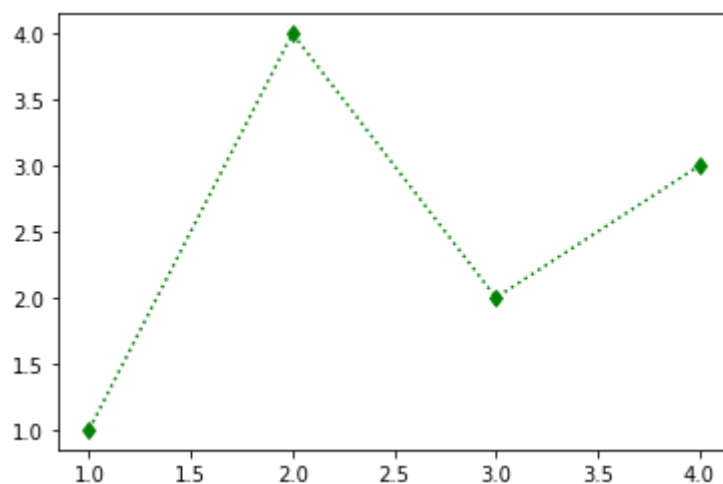
Plotting line-styles and symbols

We can set various linestyle and different symbols for the points to be plotted.

linestyle=':', '-.', '--' and '-' are the line styles. marker='^', '1', '2', '3', '4', '8', '>', '<', 'v', 'd', 's', 'o', '+', 'x', 'X', 'd', 'D', 'h', 'H', 'p' are some of the symbols for the points to be plotted

In [6]:

```
x = [1, 2, 3, 4]
y = [1, 4, 2, 3]
plt.plot(x, y, color = 'g', linestyle=':', marker='d')
plt.show()
```



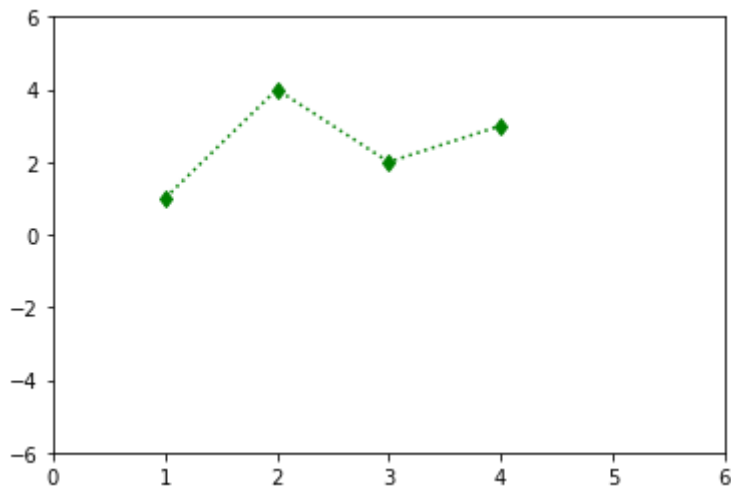
Setting the limits of the plot's axes

we use the command `plt.axis([xmin, xmax, ymin, ymax])` to set the range of the axes.

In [8]:

```
x = [1, 2, 3, 4]
y = [1, 4, 2, 3]
plt.plot(x, y, color = 'g', linestyle=':', marker='d')
```

```
plt.axis([0, 6, -6, 6]) # [xmin, xmax, ymin, ymax]
plt.show()
```



In [15]:

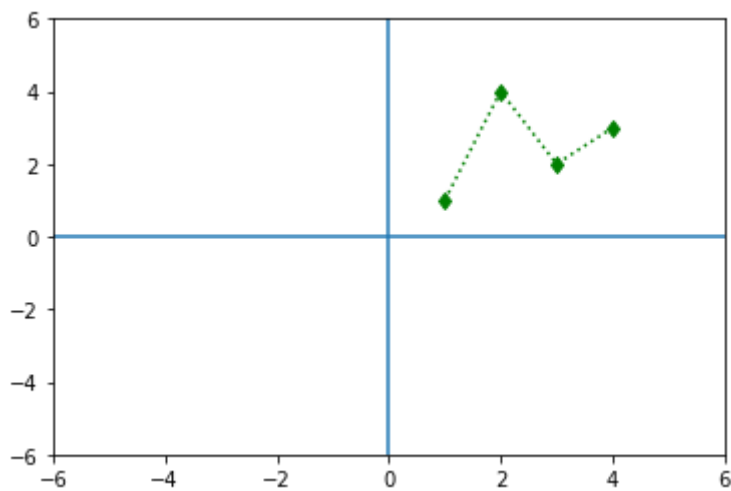
```
x = [1, 2, 3, 4]
y = [1, 4, 2, 3]

plt.plot(x, y, color = 'g', linestyle=':', marker='d')

plt.axis([-6, 6, -6, 6]) # [xmin, xmax, ymin, ymax]

plt.axhline()
plt.axvline()

plt.show()
```

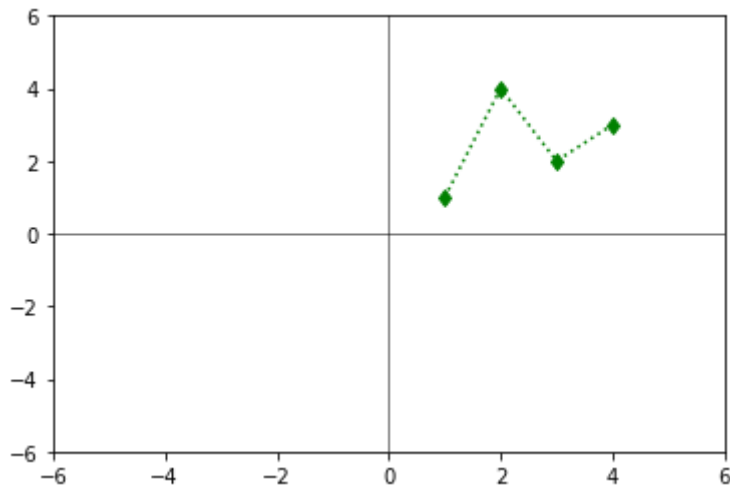


In [21]:

```
x = [1, 2, 3, 4]
y = [1, 4, 2, 3]
plt.plot(x, y, color = 'g', linestyle=':', marker='d')
plt.axis([-6, 6, -6, 6]) # [xmin, xmax, ymin, ymax]

plt.axhline(0, lw=0.5, color='black')
plt.axvline(0, lw=0.5, color='black')

plt.show()
```



linspace() and arange()

numpy.linspace return points which are equally spaced with respect to the given interval and the number of points to be returned is given as a parameter. numpy.arange return any number of points between an interval by the given step size.

In [22]: `import numpy as np`

Write a program to display 100 values between 0 and 5.

In [26]: `print("\n*****")
x=np.linspace(0,5,100)
print(x)
print("\n*****")`

```
*****
[0.          0.05050505 0.1010101  0.15151515 0.2020202  0.25252525
 0.3030303  0.35353535 0.4040404  0.45454545 0.50505051 0.55555556
 0.60606061 0.65656566 0.70707071 0.75757576 0.80808081 0.85858586
 0.90909091 0.95959596 1.01010101 1.06060606 1.11111111 1.16161616
 1.21212121 1.26262626 1.31313131 1.36363636 1.41414141 1.46464646
 1.51515152 1.56565657 1.61616162 1.66666667 1.71717172 1.76767677
 1.81818182 1.86868687 1.91919192 1.96969697 2.02020202 2.07070707
 2.12121212 2.17171717 2.22222222 2.27272727 2.32323232 2.37373737
 2.42424242 2.47474747 2.52525253 2.57575758 2.62626263 2.67676768
 2.72727273 2.77777778 2.82828283 2.87878788 2.92929293 2.97979798
 3.03030303 3.08080808 3.13131313 3.18181818 3.23232323 3.28282828
 3.33333333 3.38383838 3.43434343 3.48484848 3.53535354 3.58585859
 3.63636364 3.68686869 3.73737374 3.78787879 3.83838384 3.88888889
 3.93939394 3.98989899 4.04040404 4.09090909 4.14141414 4.19191919
 4.24242424 4.29292929 4.34343434 4.39393939 4.44444444 4.49494949
 4.54545455 4.5959596  4.64646465 4.6969697  4.74747475 4.7979798
 4.84848485 4.8989899  4.94949495 5.          ]
*****
```

Write a program to display points at a step lenght 0.5 from 0 to 5

In [28]: `print("\n*****")
y=np.arange(0,5,0.5)
print(y)
print("\n*****")`

[0. 0.5 1. 1.5 2. 2.5 3. 3.5 4. 4.5]

Write a program to display the points from 0 to 1 in descending order at a step length of 0.1.

In [29]:

```
print("\n*****")
y=np.arange(1,0,-0.1) # starts from 0 but last point be less than the end point
print(y)
print("\n*****")
```

[1. 0.9 0.8 0.7 0.6 0.5 0.4 0.3 0.2 0.1]

Evaluate $f(x) = x^2$ for 100 points in (0, 2).

In [31]:

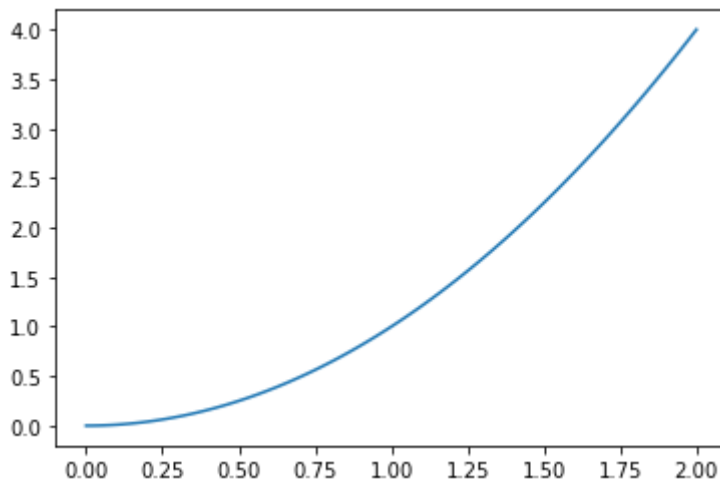
```
x = np.linspace(0, 2, 100)
y = x**2
print(y)
```

```
[0.00000000e+00 4.08121620e-04 1.63248648e-03 3.67309458e-03
 6.52994592e-03 1.02030405e-02 1.46923783e-02 1.99979594e-02
 2.61197837e-02 3.30578512e-02 4.08121620e-02 4.93827160e-02
 5.87695133e-02 6.89725538e-02 7.99918376e-02 9.18273646e-02
 1.04479135e-01 1.17947148e-01 1.32231405e-01 1.47331905e-01
 1.63248648e-01 1.79981635e-01 1.97530864e-01 2.15896337e-01
 2.35078053e-01 2.55076013e-01 2.75890215e-01 2.97520661e-01
 3.19967350e-01 3.43230283e-01 3.67309458e-01 3.92204877e-01
 4.17916539e-01 4.44444444e-01 4.71788593e-01 4.99948985e-01
 5.28925620e-01 5.58718498e-01 5.89327620e-01 6.20752984e-01
 6.52994592e-01 6.86052444e-01 7.19926538e-01 7.54616876e-01
 7.90123457e-01 8.26446281e-01 8.63585348e-01 9.01540659e-01
 9.40312213e-01 9.79900010e-01 1.02030405e+00 1.06152433e+00
 1.10356086e+00 1.14641363e+00 1.19008264e+00 1.23456790e+00
 1.27986940e+00 1.32598714e+00 1.37292113e+00 1.42067136e+00
 1.46923783e+00 1.51862055e+00 1.56881951e+00 1.61983471e+00
 1.67166616e+00 1.72431385e+00 1.77777778e+00 1.83205795e+00
 1.88715437e+00 1.94306703e+00 1.99979594e+00 2.05734109e+00
 2.11570248e+00 2.17488011e+00 2.23487399e+00 2.29568411e+00
 2.35731048e+00 2.41975309e+00 2.48301194e+00 2.54708703e+00
 2.61197837e+00 2.67768595e+00 2.74420977e+00 2.81154984e+00
 2.87970615e+00 2.94867871e+00 3.01846750e+00 3.08907254e+00
 3.16049383e+00 3.23273135e+00 3.30578512e+00 3.37965514e+00
 3.45434139e+00 3.52984389e+00 3.60616264e+00 3.68329762e+00
 3.76124885e+00 3.84001632e+00 3.91960004e+00 4.00000000e+00]
```

Plotting a function

In [32]:

```
#plot the function f(x)=x^2.
x = np.linspace(0, 2, 100)
plt.plot(x,x**2)
plt.show()
```



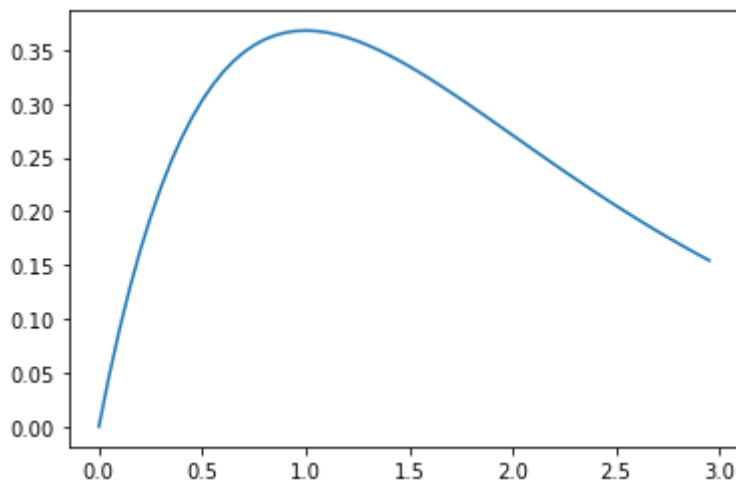
In [5]:

```
#plot  $f(x) = x * \exp(-x)$ .

def f(x):
    return x*np.exp(-x)
x = np.arange ( start = 0.,
                , stop = 3.
                , step = 0.05
                ) # generate points between start and stop with distances of step ap

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

Out[5]: <function matplotlib.pyplot.show(close=None, block=None)>



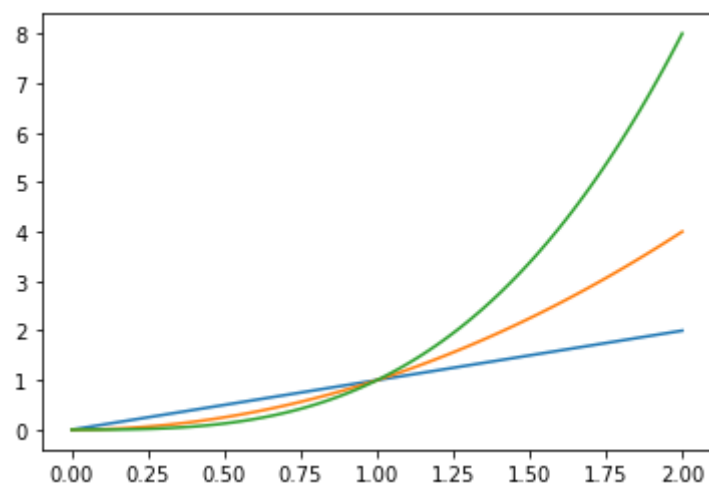
More than one plot in the same figure

the pyplot-style:

In [11]:

```
#plot the funnctions  $f(x) = x, x^2, x^3$ .

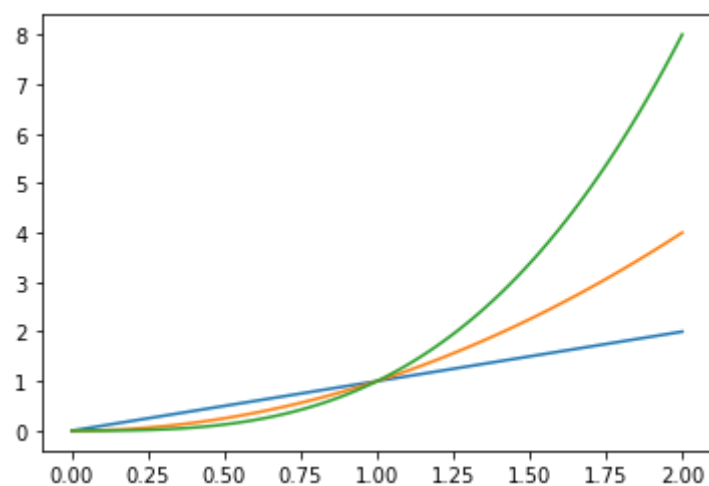
x = np.linspace(0, 2, 100)
plt.plot(x,x)
plt.plot(x,x**2)
plt.plot(x,x**3)
plt.show()
```



the OO-style:

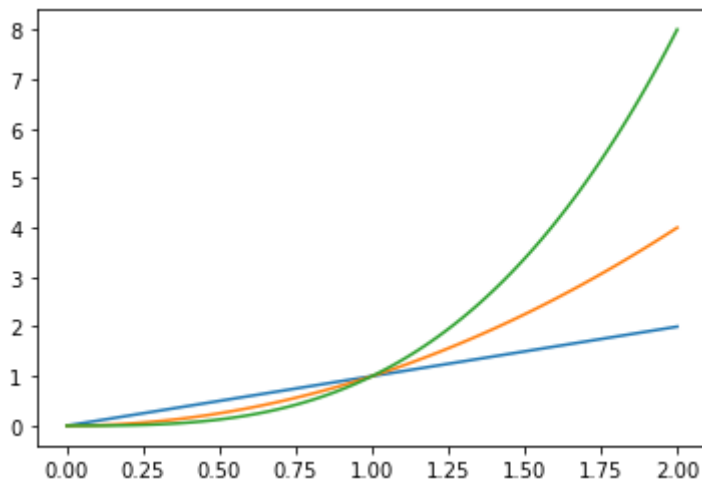
In [19]:

```
fig = plt.figure()
ax = plt.axes()
ax.plot(x, x) # Plot some data on the axes.
ax.plot(x, x**2) # Plot more data on the axes...
ax.plot(x, x**3)
plt.show()
```



In [21]:

```
# using the command subplots().
fig, ax = plt.subplots()
ax.plot(x, x) # Plot some data on the axes.
ax.plot(x, x**2) # Plot more data on the axes...
ax.plot(x, x**3) # ... and some more.
plt.show()
```



In [6]:

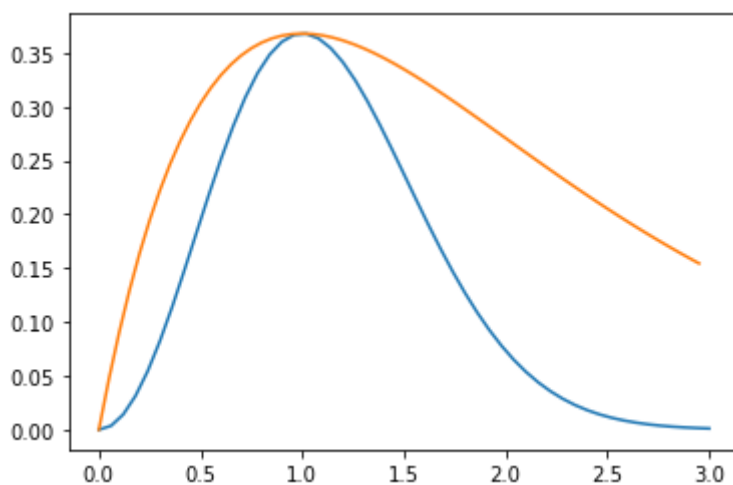
```
#plot the functions  $f(x) = x^2 * \exp(-x^2)$  and  $g(x) = x * \exp(-x)$ .

def f(x):
    return x**2*np.exp(-x**2)
x = np.linspace ( start = 0.      # lower limit
                  , stop = 3      # upper limit
                  , num = 51      # generate 51 points between 0 and 3
                  )

def g(x):
    return x*np.exp(-x)
xx = np.arange ( start = 0.
                 , stop = 3.
                 , step = 0.05
                 ) # generate points between start and stop with distances of step ap

y = f(x)
yy = g(xx)

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

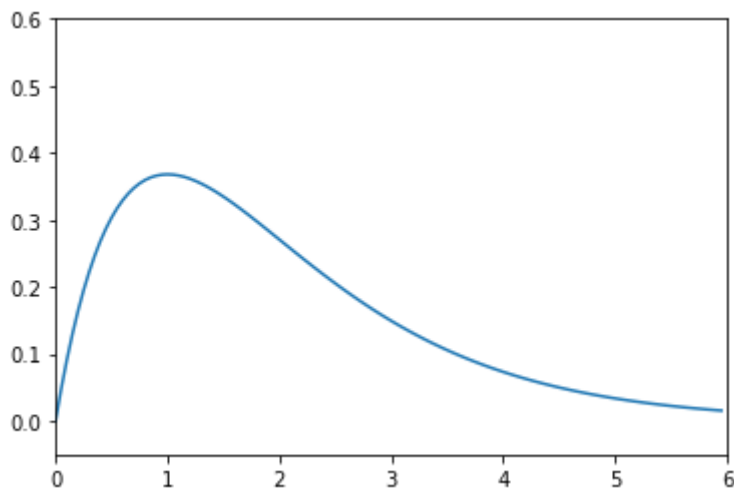


In [22]:

```
xx = np.arange ( start = 0.
                 , stop = 6.
                 , step = 0.05
                 ) # generate points between start and stop with distances of step ap
```

```
plt.plot( xx, g(xx))
plt.axis([0, 6, -0.05, 0.6]) # [xmin, xmax, ymin, ymax]
```

Out[22]: (0.0, 6.0, -0.05, 0.6)



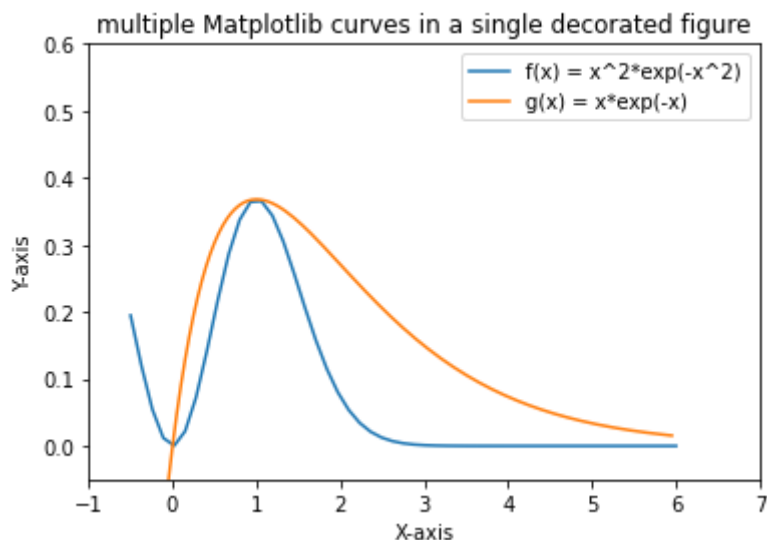
In [29]:

```
def f(x):
    return x**2*np.exp(-x**2)
x = np.linspace ( start = -0.5      # lower limit
                  , stop = 6        # upper limit
                  , num = 51        # generate 51 points between 0 and 3
                  )

def g(x):
    return x*np.exp(-x)
xx = np.arange ( start = -0.5
                 , stop = 6.
                 , step = 0.05
                 ) # generate points between start and stop with distances of step ap

y = f(x)
yy = g(xx)

plt.plot(x,y)
plt.plot( xx, yy)
plt.axis([-1, 7, -0.05, 0.6])
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.legend( [ 'f(x) = x^2*exp(-x^2)' # This is f(x)
             , 'g(x) = x*exp(-x)'   # This is g(x)
             ] )
plt.title('multiple Matplotlib curves in a single decorated figure')
plt.show()
```

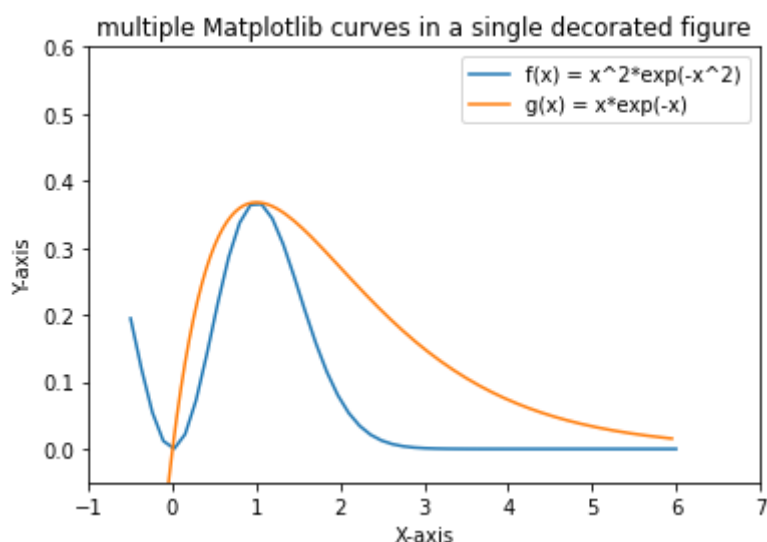


Saving figures as external files

In [33]:

```
y = f(x)
yy = g(xx)

plt.plot(x,y)
plt.plot( xx, yy)
plt.axis([-1, 7, -0.05, 0.6])
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.legend( [ 'f(x) = x^2*exp(-x^2)' # This is f(x)
             , 'g(x) = x*exp(-x)'   # This is g(x)
           ] )
plt.title('multiple Matplotlib curves in a single decorated figure')
plt.savefig('multipleCurvesFullRangeDecorated.pdf') # produces a PDF file containing
plt.savefig('multipleCurvesFullRangeDecorated.png') # produces a PNG file containing
plt.show()
```



In [48]:

```
x = np.linspace ( start = -0.5 # lower limit
                  , stop = 6    # upper limit
                  , num = 51
                  )

xx = np.arange ( start = -0.5
                , stop = 6.
                )
```

```

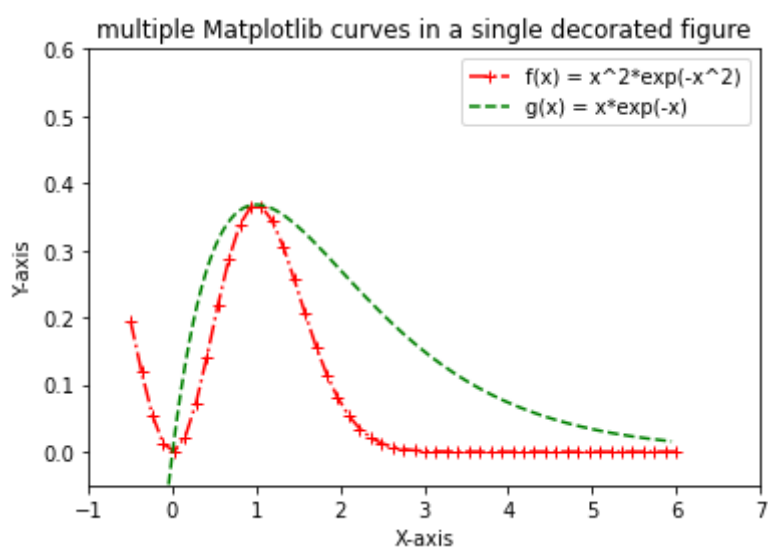
        , step = 0.05
    ) # generate points between start and stop with distances of step ap

y = f(x)
yy = g(xx)

plt.plot(x,y, 'r', linestyle='-.')
plt.plot( xx, yy, 'g', linestyle='--')
plt.axis([-1, 7, -0.05, 0.6])
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.legend( [ 'f(x) = x^2*exp(-x^2)'    # This is f(x)
             , 'g(x) = x*exp(-x)'      # This is g(x)
             ] )
plt.title('multiple Matplotlib curves in a single decorated figure')

plt.show()

```



Scatter Plot

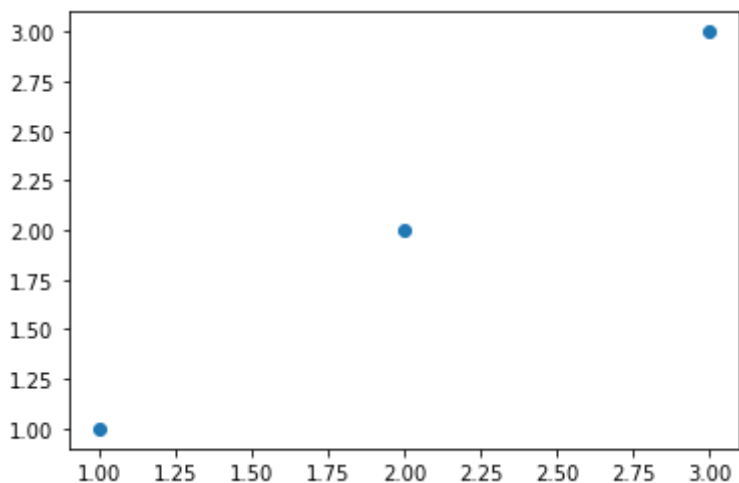
The command 'scatter()' is used to obtain a scatter plot.

In [55]:

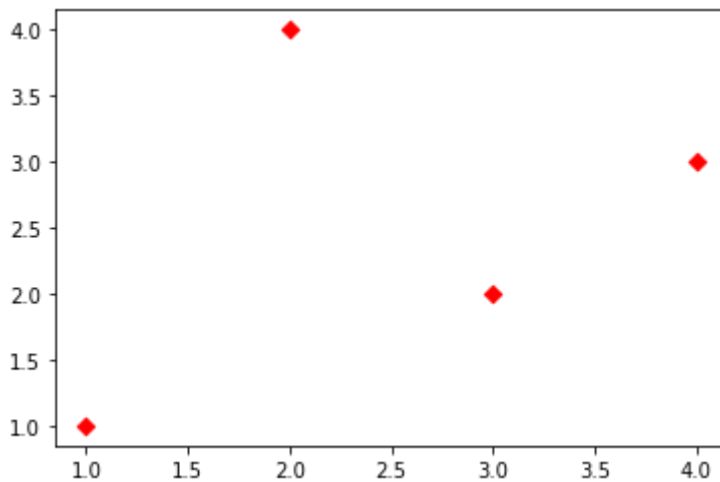
```

x = [1, 2, 3]
y = [1, 2, 3]
plt.scatter(x, y)
plt.show()

```

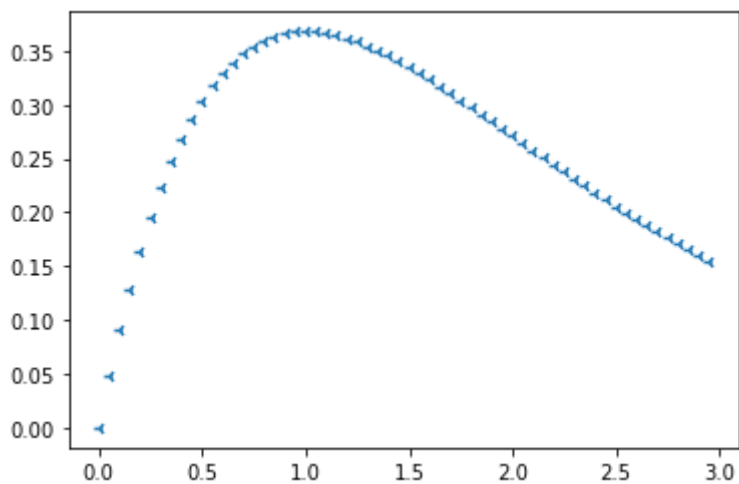


```
In [58]: x = [1, 2, 3, 4]
y = [1, 4, 2, 3]
plt.scatter(x, y, color = 'r', marker='D')
plt.show()
```



```
In [60]: def f(x):
return x*np.exp(-x)
x = np.arange ( start = 0.
, stop = 3.
, step = 0.05
) # generate points between start and stop with distances of step ap
y = f(x)
plt.scatter( x, y, marker='3')
plt.show
```

```
Out[60]: <function matplotlib.pyplot.show(close=None, block=None)>
```



```
In [64]: def f(x):
return x**2*np.exp(-x**2)
x = np.linspace ( start = -0.5 # Lower Limit
, stop = 6 # upper limit
, num = 51 # generate 51 points between 0 and 3
)

def g(x):
return x*np.exp(-x)
xx = np.arange ( start = -0.5
, stop = 6.
, step = 0.05
```

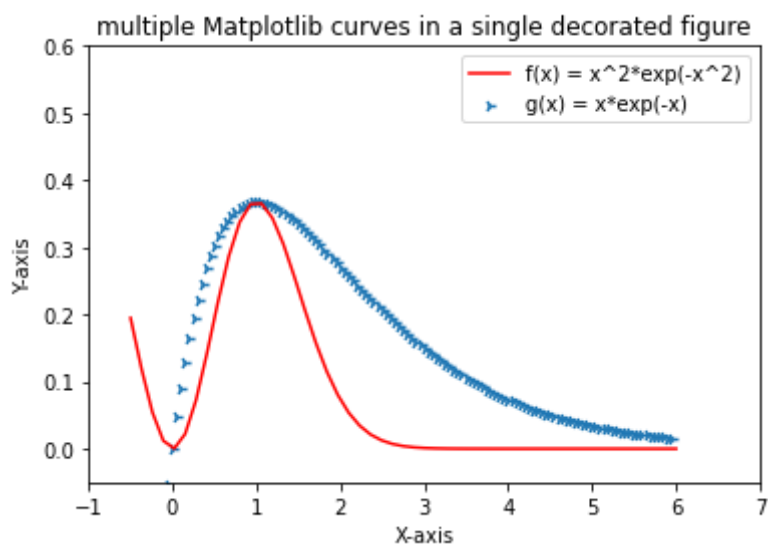
```

    ) # generate points between start and stop with distances of step ap

y = f(x)
yy = g(xx)

plt.plot(x,y, color='r')
plt.scatter( xx, yy, marker = '4')
plt.axis([-1, 7, -0.05, 0.6])
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.legend( [ 'f(x) = x^2*exp(-x^2)'    # This is f(x)
             , 'g(x) = x*exp(-x)'      # This is g(x)
             ] )
plt.title('multiple Matplotlib curves in a single decorated figure')
plt.show()

```



Vector Fields

Plot the vector field $\vec{F}(x, y) = i - j$.

In [2]:

```

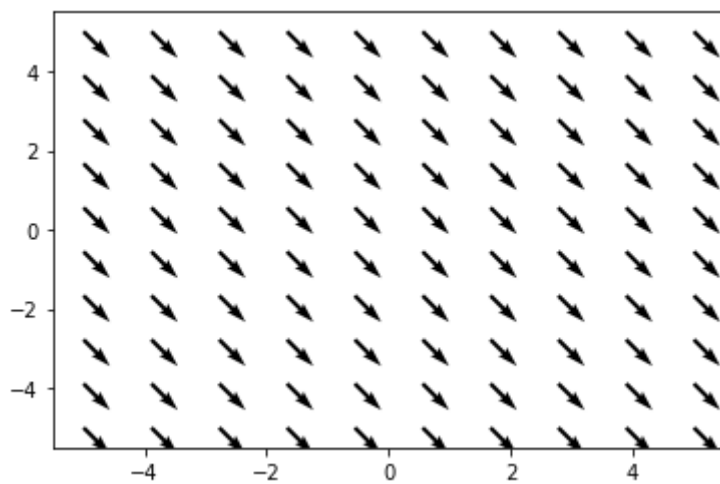
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

x,y = np.meshgrid(np.linspace(-5,5,10),np.linspace(-5,5,10))

u = 1
v = -1

plt.quiver(x,y,u,v)
plt.show()

```



Plot the vector field $\vec{F}(x, y) = \frac{x}{\sqrt{x^2+y^2}}\mathbf{i} + \frac{y}{\sqrt{x^2+y^2}}\mathbf{j}$

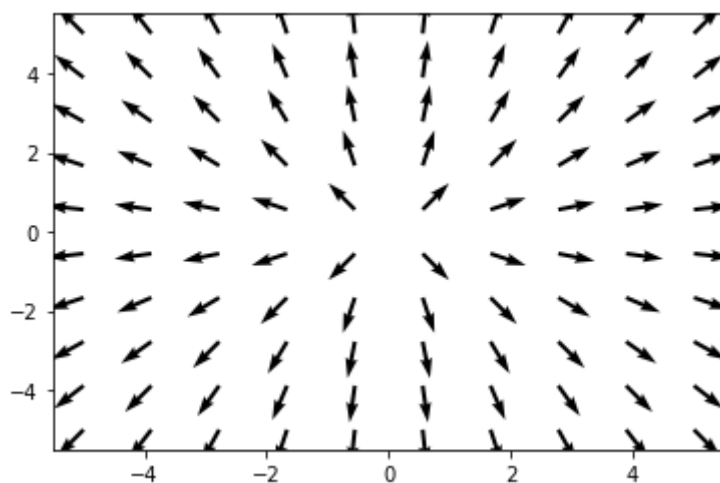
In [3]:

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

x,y = np.meshgrid(np.linspace(-5,5,10),np.linspace(-5,5,10))

u = x/np.sqrt(x**2 + y**2)
v = y/np.sqrt(x**2 + y**2)

plt.quiver(x,y,u,v)
plt.show()
```



Plot the vector field $\vec{F}(x, y) = -\frac{y}{\sqrt{x^2+y^2}}\mathbf{i} + \frac{x}{\sqrt{x^2+y^2}}\mathbf{j}$

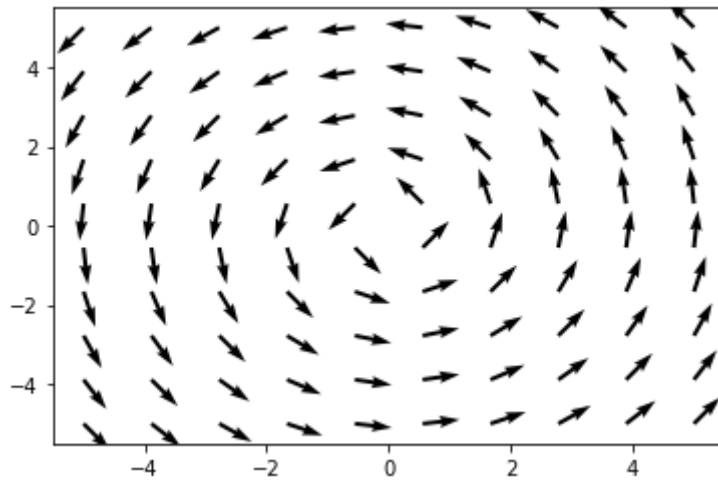
In [4]:

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

x,y = np.meshgrid(np.linspace(-5,5,10),np.linspace(-5,5,10))

u = -y/np.sqrt(x**2 + y**2)
v = x/np.sqrt(x**2 + y**2)

plt.quiver(x,y,u,v)
plt.show()
```



Visualizing data from CSV file in Python

CSV stands for 'Comma-Separated Values'. It means the data(values) in a CSV file are separated by a delimiter i.e., comma. Data in a CSV file is stored in tabular format with an extension of .csv. Generally, CSV files are used with Google spreadsheets or Microsoft Excel sheets. A CSV file contains a number of records with the data spread across rows and columns. We are going to visualize data from a CSV file in Python.

To extract the data in CSV file, CSV module must be imported in our program as follows:

In [4]:

```
import csv
rows=[]
with open('biostats.csv') as File:
    Line_reader = csv.reader(File)
    header=next(Line_reader)
    for row in Line_reader:
        rows.append(row)

print(header)
print(rows)
```

```
['i»;Name', 'Sex', 'Age']
[['Alwin', 'M', '40'], ['Brian', 'M', '42'], ['Collin', 'M', '32'], ['Derick', 'M', '35'], ['Emily', 'F', '30'], ['Fathima', 'F', '31'], ['Gunther', 'M', '26'], ['Harry', 'M', '28'], ['Ishan', 'M', '50']]
```

In [11]:

```
%matplotlib inline
import matplotlib.pyplot as plt
import csv

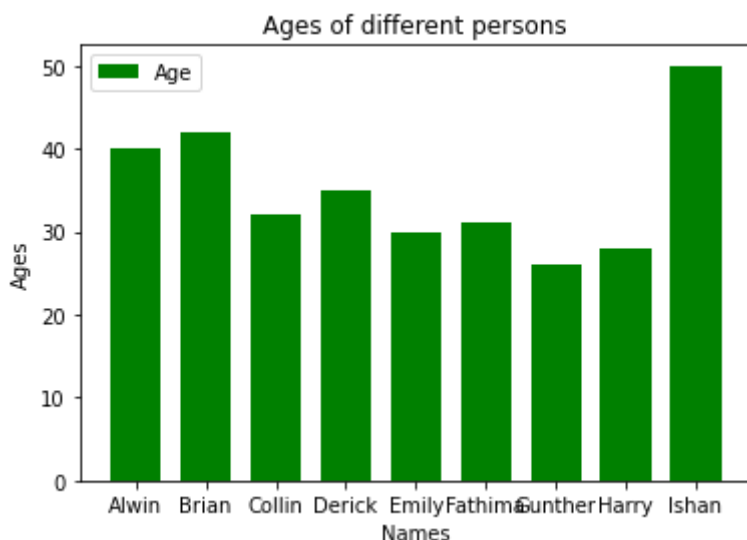
x = []
y = []

with open('biostats.csv','r') as csvfile:
    plots = csv.reader(csvfile, delimiter = ',')

    for row in plots:
        x.append(row[0])
        y.append(int(row[2]))

plt.bar(x, y, color = 'g', width = 0.72, label = "Age")
plt.xlabel('Names')
plt.ylabel('Ages')
plt.title('Ages of different persons')
```

```
plt.legend()
plt.show()
```



In [3]:

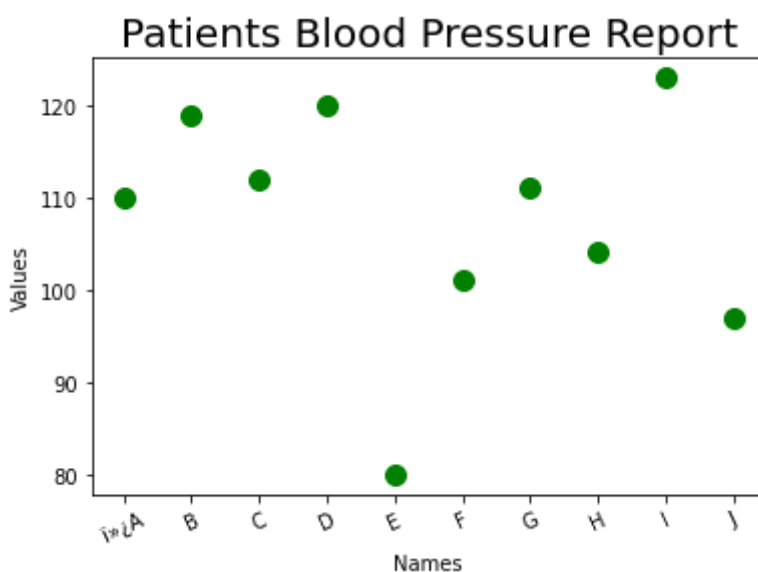
```
import matplotlib.pyplot as plt
import csv

Names = []
Values = []

with open('bldprs_measure.csv','r') as csvfile:
    lines = csv.reader(csvfile, delimiter=',')
    for row in lines:
        Names.append(row[0])
        Values.append(int(row[1]))

plt.scatter(Names, Values, color = 'g',s = 100)
plt.xticks(rotation = 25)
plt.xlabel('Names')
plt.ylabel('Values')
plt.title('Patients Blood Pressure Report', fontsize = 20)

plt.show()
```



In [4]:

```
import matplotlib.pyplot as plt
import csv
```

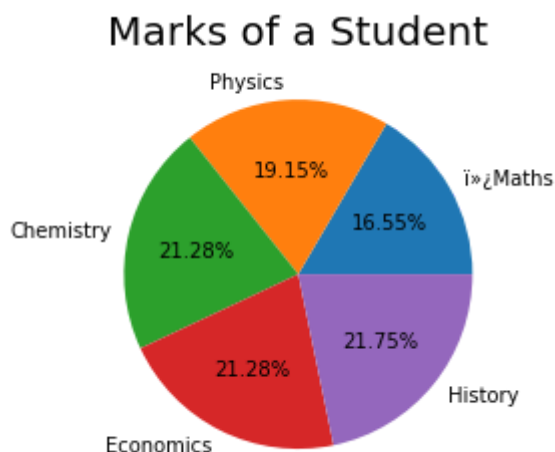
```

Subjects = []
Scores = []

with open('SubjectMarks.csv', 'r') as csvfile:
    lines = csv.reader(csvfile, delimiter = ',')
    for row in lines:
        Subjects.append(row[0])
        Scores.append(int(row[1]))

plt.pie(Scores, labels = Subjects, autopct = '%.2f%')
plt.title('Marks of a Student', fontsize = 20)
plt.show()

```



In [5]:

```

import matplotlib.pyplot as plt
import csv

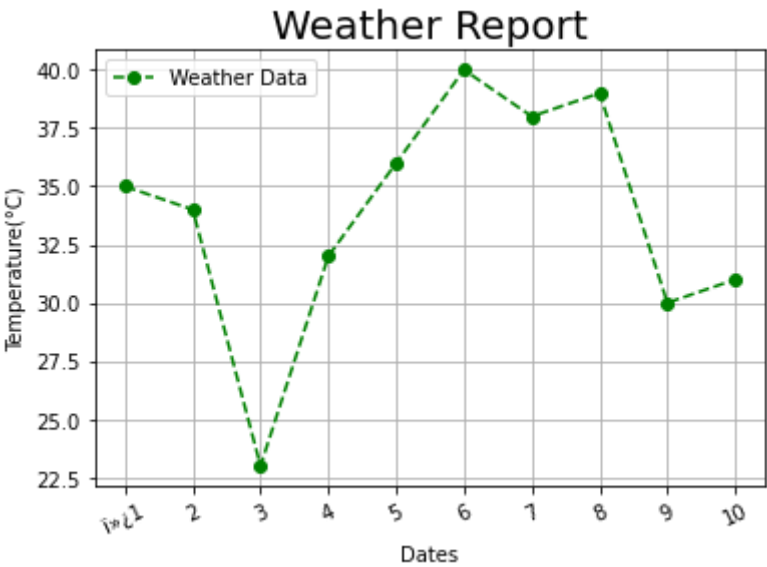
x = []
y = []

with open('Weatherdata.csv','r') as csvfile:
    lines = csv.reader(csvfile, delimiter=',')
    for row in lines:
        x.append(row[0])
        y.append(int(row[1]))

plt.plot(x, y, color = 'g', linestyle = 'dashed',
         marker = 'o', label = "Weather Data")

plt.xticks(rotation = 25)
plt.xlabel('Dates')
plt.ylabel('Temperature(°C)')
plt.title('Weather Report', fontsize = 20)
plt.grid()
plt.legend()
plt.show()

```



In []: