Workshop with Hands-on Training on Python for Physics

Key topics Covered

1. Introduction to Jupiter Notebook
2. Introduction to Basics of Python
3. Python libraries (Numpy and Matplotlib)
4. Programming the main topics of Physics:
    - Vectors
    - Motion & Free Fall Motion
    - Projectile motion
    - Simple Harmonic Motion & Damped Oscillation
    - Circular Motion &SHM
    - Wave Motion
    - Electrostatics Force and Field
    - Gravitational Field

Introduction to Python

Python as a Calculator

In [598]:
```
3+4
```

Out[598]: 7

In [599]:
```
2+4 , 3-5, 6/2 , 5*3
```

Out[599]: $(6, \quad -2, \quad 3.0, \quad 15)$

In [600]:
```
5**3
```

Out[600]: 125

Common types in Python

- **Numeric** : Intergers , floats , complex
- **Sequence** : List , tuple , range
- **Binary** : Byte , bytearray
- **True/ Flase** : bool
- **Text** : String

Variable

In [601]:
```
a = 4
b= 3.5
c = 'Physics'
list = [1,2,3,4]
print (a,'  ,', b , ',', c , ',' ,list)
```

```
4   , 3.5 , Physics , [1, 2, 3, 4]
```

In [602]:
```
print (type(a), type(b), type(c), type(list))
```

```
<class 'int'> <class 'float'> <class 'str'> <class 'list'>
```

In [603]:
```
print ("Hello World")
print(a)
print(" The course name:   ", c)
```

```
Hello World
```

```
4
 The course name:   Physics
```

String Operations

In [606]:
```
s1 = "applied"
s2 = "Physics"
s1+s2
```

Out[606]: 'appliedPhysics'

In [607]:
```
 s1.capitalize()
```

Out[607]: 'Applied'

In [608]:
```
L = "PAKISTAN"
L.lower()
```

Out[608]: 'pakistan'

In [609]:
```
L*3
```

Out[609]: 'PAKISTANPAKISTANPAKISTAN'

In [611]:
```
sp = 'Lets split the word'
sp.split('s ')
```

Out[611]: ['Let', 'split the word']

In [612]:
```
spac = 'English, Urdu, French'
spac.split(', ')
```

Out[612]: ['English', 'Urdu', 'French']

In [613]:
```
print(s1 + " " + s2) # for space b/w s1 and s2
```

```
applied Physics
```

In [614]:
```
s1[0] , s1[1]
```
Out[614]:
```
('a', 'p')
```

In [616]:
```
s1[0:4]
```
Out[616]:
```
'appl'
```

In [617]:
```
s1[3:]
```
Out[617]:
```
'lied'
```

In [618]:
```
s1[0::+3]
```
Out[618]:
```
'ald'
```

In [619]:
```
s2[0::+2]
```
Out[619]:
```
'Pyis'
```

In [620]:
```
s1[2::-1] , s2[::-1]
```
Out[620]:
```
('ppa', 'scisyhP')
```

In [430]:
```
s1 ,s2 , s3
```
Out[430]:
```
('Applied', 'Physics', 'Applied')
```

Boolean data Type

In [621]:
```
s3 = 'Applied'
s1 == s2 , s1 == s3 , s2 ==s3
```
Out[621]:
```
(False, False, False)
```

In [622]:
```
b1 = True
b2 = False
type(b1) , type(b2)
```

Out[622]: (bool, bool)

In [623]:
```
zero_int = 0  #An int, float or complex number set to zero returns as F
alse. An integer,
                  #float or complex number set to any other number, pos
itive or negative, returns as True.
bool(zero_int)
```

Out[623]: False

In [624]:
```
pos_int = 1
f = -0
neg = -2.3
bool(pos_int) , bool(s1) , bool(b1), bool(b2), bool(f), bool(neg)
```

Out[624]: (True, True, True, False, False, True)

In [625]:
```
f = 0.0
fr = 0.22
bool(f) , bool(fr)
```

Out[625]: (False, True)

In [626]:
```
name = "Anaya"
empty = ""

bool(name),  bool(empty)
```

Out[626]: (True, False)

In [627]:
```
b1 or b2
```

Out[627]: True

```
In [628]: b1 and b2
Out[628]: False
```

```
In [629]: not b1
Out[629]: False
```

```
In [630]: b1 == b2
Out[630]: False
```

```
In [631]: b1 != b2
Out[631]: True
```

List

```
In [632]: list1 = ["physics", "Chemistry", "Math", "Statistics"]   # indexing str
          at from  0 and then , 1, 2, 3
          type(list1)
Out[632]: list
```

```
In [633]: type(list1[2])
Out[633]: str
```

```
In [634]: list1[0]
Out[634]: 'physics'
```

```
In [635]: list1[1]
Out[635]: 'Chemistry'
```

In [636]:
```
list1
```
Out[636]:
```
['physics', 'Chemistry', 'Math', 'Statistics']
```

Lists are mutable

In [639]:
```
list1[2] = 22
```

In [640]:
```
list1
```
Out[640]:
```
['physics', 'Chemistry', 22, 'Statistics']
```

In [641]:
```
type(list1[2])
```
Out[641]:
```
int
```

In [642]:
```
list1
```
Out[642]:
```
['physics', 'Chemistry', 22, 'Statistics']
```

Appending to a list using " append and extend"

In [643]:
```
list2 = [1,2,3]
list1.extend(list2)
list1
```
Out[643]:
```
['physics', 'Chemistry', 22, 'Statistics', 1, 2, 3]
```

In [644]:
```
list1.append(list2)
list1
```
Out[644]:
```
['physics', 'Chemistry', 22, 'Statistics', 1, 2, 3, [1, 2, 3]]
```

In [645]:
```
list1.extend("ITC")
print(list1)
```

```
['physics', 'Chemistry', 22, 'Statistics', 1, 2, 3, [1, 2, 3], 'I',
'T', 'C']
```

In [646]:
```
list1.append("Islamiat")
print (list1)
```

```
['physics', 'Chemistry', 22, 'Statistics', 1, 2, 3, [1, 2, 3], 'I',
'T', 'C', 'Islamiat']
```

In [647]:
```
list1.append(list2)
print (list1)
```

```
['physics', 'Chemistry', 22, 'Statistics', 1, 2, 3, [1, 2, 3], 'I',
'T', 'C', 'Islamiat', [1, 2, 3]]
```

In [648]:
```
type(list1[-1])
```

Out[648]:
```
list
```

Deleting from a list using " remove and pop "

In [654]:
```
list1.remove(22)
list1
```

Out[654]:
```
['physics', 'Statistics', 3, 'I', 'T', 'C', 'Islamiat']
```

In [655]:
```
list1.pop(1)
```

Out[655]:
```
'Statistics'
```

In [656]:
```
list1
```

Out[656]: ['physics', 3, 'I', 'T', 'C', 'Islamiat']

Tuples in Python

Tuples are immutable

In [657]:
```
tuple1 = ('AP', 'PF','Eng')
tuple1
```

Out[657]: ('AP', 'PF', 'Eng')

In [658]:
```
tuple1[2]
```

Out[658]: 'Eng'

In [659]:
```
tuple1[0] = "CP"
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call l
ast)
<ipython-input-659-21558ea95455> in <module>()
----> 1 tuple1[0] = "CP"

TypeError: 'tuple' object does not support item assignment
```

Binary

In [660]:
```
dec = 320
print("The decimal value of", dec, "is:", dec)
print(bin(dec), "in binary.")
print(oct(dec), "in octal.")
print(hex(dec), "in hexadecimal.")
```

```
The decimal value of 320 is: 320
0b101000000 in binary.
0o500 in octal.
0x140 in hexadecimal.
```

In [663]:
```
bin(5), hex(10)
```

Out[663]:
```
('0b101', '0xa')
```

Loop

range(start, stop [, step])

In [664]:
```
for i in range(0,5):
    print(i)
```

```
0
1
2
3
4
```

In [666]:
```
for i in range(0,18,4):
    print(i)
```

```
0
4
8
12
16
```

In [667]:
```
i = 0
while i<10:
    print (i)
    i+= 2
```

```
0
2
4
6
8
```

Importing module

To use Python's trig functions, we need to introduce a new concept: importing modules

import mudule

from mudule import function

In [668]:
```
import math
sinx = math.sin(60)
sinx
```

Out[668]: -0.3048106211022167

In [669]:
```
math.sin(60)
```

Out[669]: -0.3048106211022167

In [670]:
```
math.e
```

Out[670]: 2.718281828459045

In [671]:
```
x= 45
math.sin(x), math.cos(x), math.tan(x)
```

Out[671]: $(0.8509035245341184, \quad 0.5253219888177297, \quad 1.6197751905438615)$

In [673]:
```
math.sin(math.radians(45))
```
Out[673]: 0.7071067811865476

In [674]:
```
math.e , math.log(x) , math.exp(x)
```
Out[674]: $(2.718281828459045, \quad 3.8066624897703196, \quad 3.4934271057485095e+19)$

In [675]:
```
math.pow(3,2) , math.pow(x,2)
```
Out[675]: $(9.0, \quad 2025.0)$

In [676]:
```
math.sqrt(x)
```
Out[676]: 6.708203932499369

In [677]:
```
math.sqrt(2**2 + 3**2)
```
Out[677]: 3.605551275463989

In [679]:
```
math.cos(math.radians(45))
```
Out[679]: 0.7071067811865476

Define a Function

In [472]:
```
def force(m,a):
    f=m*a
    return f


force(1,52)
```
Out[472]: 52

In [681]:
```
def force(a):
    f= 3*a
    print (f)
force(4)
```

12

In [685]:
```
force(4)
```

12

In [683]:
```
def abs_value(x):
    if x < 0:
        return 0-x
    return x

abs_value(-45), abs_value(4)
```

Out[683]: $(45, \quad 4)$

In [686]:
```
abs_value(-5)
```

Out[686]: 5

In [687]:
```
import numpy as np
import math

def Magnitude(ax,ay):
    a = np.sqrt(ax**2 + ay**2)
    return a
Magnitude(7, 3)
```

Out[687]: 7.615773105863909

In [688]:
```
def components(mag,theta):
    a=math.radians(theta)
    ax = mag*math.cos(a)
```

```
    ay = mag*math.sin(a)
    print("Ax:",ax)
    return ax, ay
components(4,45)
```

```
Ax: 2.8284271247461903
```

Out[688]: $(2.8284271247461903, \quad 2.8284271247461903)$

In [689]:
```
def angle(x,y):
    ang = np.arctan(y/x)
    a = np.degrees(ang)
    return a
angle(3,3)
```

Out[689]: 45.0

Numpy Library

Numerical Python, or **"Numpy"** for short, is a foundational package on which many of the most common data science packages are built. Numpy provides us with high performance multi-dimensional arrays which we can use as vectors or matrices. The key features of numpy are:

1. **ndarrays**: n-dimensional arrays of the same data type which are fast and space-efficient. There are a number of built-in methods for ndarrays which allow for rapid processing of data without using loops (e.g., compute the mean).
2. **Broadcasting**: a useful tool which defines implicit behavior between multi-dimensional arrays of different sizes.
3. **Vectorization**: enables numeric operations on ndarrays.
4. **Input/Output** : simplifies reading and writing of data from/to file.

In [690]:
```
import numpy as np

an_array = np.array([3, 33, 333])  # Create a rank 1 array
```

```
print(type(an_array))            # The type of an ndarray is: "<class
 'numpy.ndarray'>"
```

```
<class 'numpy.ndarray'>
```

In [691]:
```
print(an_array.shape)
```

```
(3,)
```

In [692]:
```
print(an_array[0], an_array[1], an_array[2])
```

```
3 33 333
```

In [693]:
```
an_array[0] =888            # ndarrays are mutable, here we change an e
lement of the array

an_array
```

Out[693]:
```
array([888,  33, 333])
```

How to create a Rank 2 numpy array:

A rank 2 ndarray is one with two dimensions.

In [694]:
```
another = np.array([[11,12,13],[21,22,23]])   # Create a rank 2 array

print(another)  # print the array

print("The shape is 2 rows, 3 columns: ", another.shape)  # rows x colu
mns

print("Accessing elements [0,0], [0,1], and [1,0] of the ndarray: ", an
other[0, 0], ", ",another[0, 1],", ", another[1, 0])
```

```
[[11 12 13]
 [21 22 23]]
The shape is 2 rows, 3 columns:  (2, 3)
```

```
Accessing elements [0,0], [0,1], and [1,0] of the ndarray:  11 ,  12 , 
21
```

In [695]:
```python
# create a 2x2 array of zeros
ex1 = np.zeros((2,2))
print(ex1)  , ex1.shape
```

```
[[0. 0.]
 [0. 0.]]
```

Out[695]:
```
(None, (2, 2))
```

In [696]:
```python
ex3 = np.eye(3,3)
print(ex3)
```

```
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

In [697]:
```python
ex4 = np.ones((1,2))
print(ex4)
```

```
[[1. 1.]]
```

In [698]:
```python
# notice that the above ndarray (ex4) is actually rank 2, it is a 1x2 a
rray
print(ex4.shape)

# which means we need to use two indexes to access an element

print(ex4[0,0])
```

```
(1, 2)
1.0
```

In [700]:
```python
# create an array of random floats between 0 and 1
ex5 = np.random.random((4,4))
print(ex5)
```

```
[[0.52282538 0.42292659 0.33101524 0.87424826]
 [0.34352157 0.74214962 0.44966889 0.63849355]
 [0.0844776  0.26002894 0.74076949 0.45670974]
 [0.48417854 0.25535966 0.14353242 0.19095065]]
```

In [703]:
```
ex2 = np.array([11.0, 12.0]) # Python assigns the  data type
print(ex2.dtype)
```

```
float64
```

In [704]:
```
ex3 = np.array([11, 21], dtype=np.int64) #You can also tell Python the
 data type
print(ex3.dtype)
```

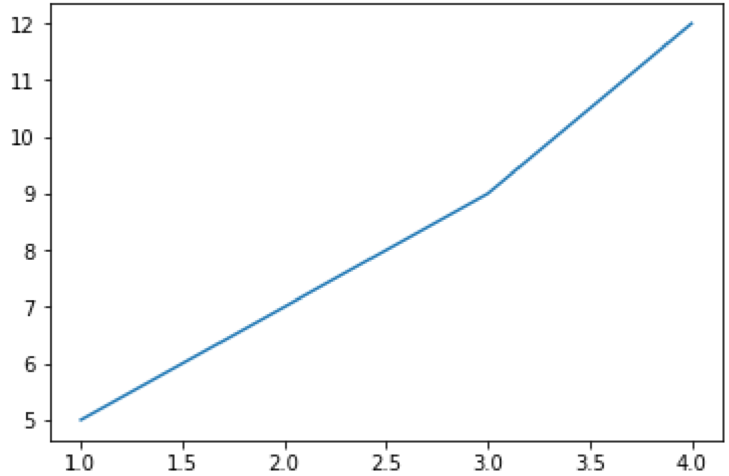
```
int64
```

In [705]:
```
# you can use this to force floats into integers (using floor function)
ex4 = np.array([11.1,12.7], dtype=np.int64)
print(ex4.dtype)
print()
print(ex4)
```

```
int64

[11 12]
```

Matplotlib

In [706]:
```
import matplotlib.pyplot as plt

x = np.array([1,2,3,4])
y = np.array([5,7,9,12])

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



Python for Physics

Vector

Magnitude of a Vector

$A = Axi + Ayj + Azk$

$A = \sqrt(Ax^2 + Ay^2)$

In [707]:
```
vector = np.array([2,4,7])
magnitude_vector = np.linalg.norm(vector)
print ("The magnitude of the vector ", vector , "is :", magnitude_vecto
r)
```

```
The magnitude of the vector  [2 4 7] is : 8.306623862918075
```

In [708]:
```
np.sqrt(2**2 + 4**2 + 7**2)
```

Out[708]: 8.306623862918075

Horizontal and Vertical Components of a Vector

$$Ax = ACosx$$

$$Ay = ASinx$$

In [709]:
```
an= 45
A = np.array([5,4])
Ax= A[0]*np.cos(an)
Ay= A[1]*np.sin(an)
Ax, Ay
```

Out[709]: $(2.626609944088649, \quad 3.4036140981364738)$

Find the angle between “a” and “b”.

where

$$a = 5i + 4j - 6k$$

$$b = -2i + 2j + 3k$$

$$A.B = ABcos\theta$$

$$\theta = cos - 1(A.B/AB)$$

In [711]:
```
import numpy as np

a =np.array([5,4,-6])
b =np.array([-2, 2,3])

# For a.b = cos theta
```

```
a_dot_b = np.dot(a,b)


mag_a = np.linalg.norm(a)  # magnitude of a
mag_b = np.linalg.norm(b)  # magnitude of b


value = (a_dot_b / (mag_a * mag_b)) # this is in radian
angle = np.arccos(value)
direction = np.degrees(angle) # radian to degree
print ('The angle between a and b is:\n',  direction)
```

```
The angle between a and b is:
 123.55862948381244
```

In [712]:
```
a*b
```

Out[712]:
```
array([-10,   8, -18])
```

In [ ]:
```

```

Angle with respect to x , y and z axes

In [713]:
```
def Angle(s):
    xcosine_angle = s[0] / (np.linalg.norm(s))
    x = np.arccos(xcosine_angle)
    anglex = np.degrees(x)

    ycosine_angle = s[1] / (np.linalg.norm(s))
    y = np.arccos(ycosine_angle)
    angley = np.degrees(y)

    zcosine_angle = s[2] / (np.linalg.norm(s))
    z = np.arccos(zcosine_angle)
    anglez = np.degrees(z)

    print('The angle with X is : ', anglex)
```

```
    print('The angle with Y is : ', angley)
    print('The angle with Z is : ', anglez)

vector =[2,-3,5]

Angle(vector)
```

```
The angle with X is :  71.06817681913482
The angle with Y is :  119.12156807035144
The angle with Z is :  35.795759914707084
```

One Dimension Kinematics

The position of a particle moving in a straight line is given by

$$X = 5 + 2t + 4t2 - t3$$

where x is in meter. (a) Find an expression for the Velocity and Acceleration as a function of time. (b) Find the position of the particle at t=1 sec

In [722]:
```
import sympy as sp
sp.init_printing()
t = sp.symbols('t')
Position = 2*t + 4*t**2 + t**3+ 5
velocity = sp.diff(Position,t)
acceleration = sp.diff(Position,t,2)
```

print('The position is : ') Position

In [723]:
```
print('The velocity is : ')
velocity
```

```
The velocity is :
```

Out[723]: $3t^2 + 8t + 2$

In [724]:
```
print('The acceleration is : ')
```

```
acceleration
```

The acceleration is :

Out[724]: $2(3t + 4)$

Define function

In [725]:
```
import math
from scipy.misc import derivative

def f(x):
    fn = math.sin(x)
    return fn

derivative(f,45, dx =0.1)
```

Out[725]: 0.5244468898338156

In [726]:
```
math.cos(45)
```

Out[726]: 0.5253219888177297

In [727]:
```
derivative(math.cos,45, dx =1e-2)
```

Out[727]: -0.8508893428794517

In [728]:
```
math.sin(45)
```

Out[728]: 0.8509035245341184

In [733]:
```
def position (t):
    x = 2*t + 4*t**2 + t**3+ 5
    return x

print('the velocity of the particle at t=1.0sec  is : ')
```

```
derivative(position,1.0, dx = 1e-5)
```

```
the velocity of the particle at t=1.0sec  is :
```

Out[733]: 13.000000000129573

Free Fall Motion

$g = \frac{F}{m}$

$v = g * t$

$h = \frac{1}{2} * g * t^2$

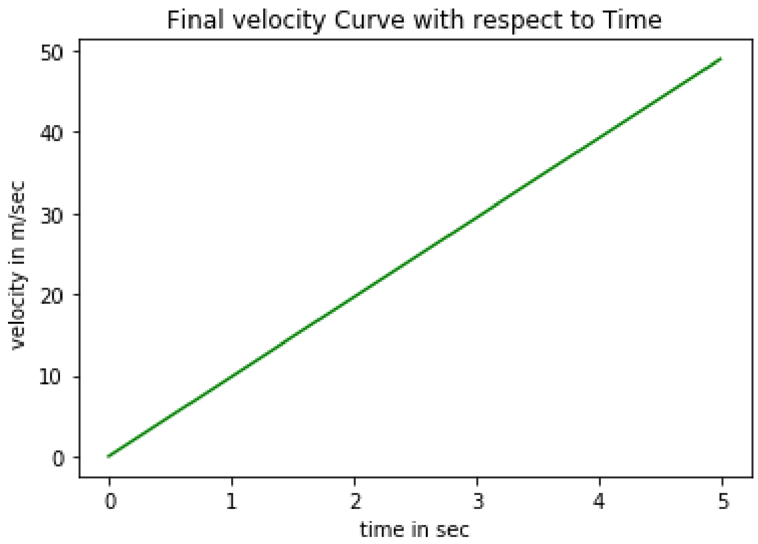
In [735]:
```
np.arange(1,10,.1)
```

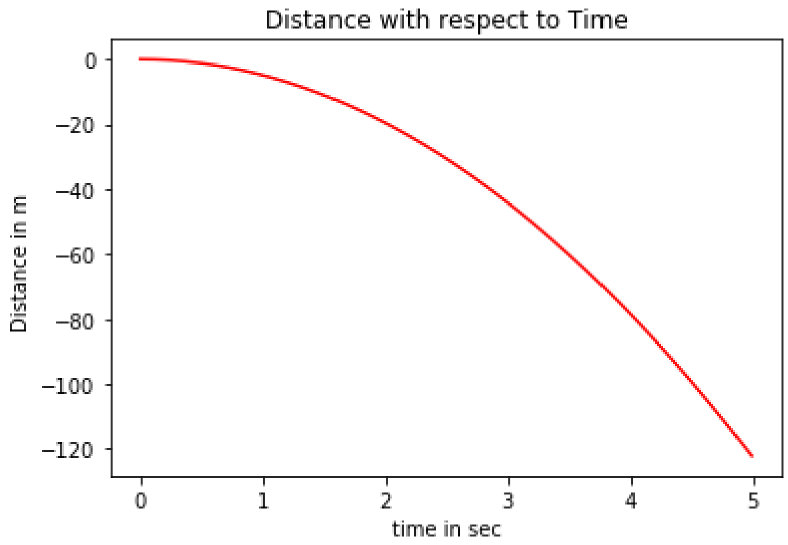
Out[735]:
```
array([1. , 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2. , 2.1, 2.2,
       2.3, 2.4, 2.5, 2.6, 2.7, 2.8, 2.9, 3. , 3.1, 3.2, 3.3, 3.4, 3.5,
       3.6, 3.7, 3.8, 3.9, 4. , 4.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.7, 4.8,
       4.9, 5. , 5.1, 5.2, 5.3, 5.4, 5.5, 5.6, 5.7, 5.8, 5.9, 6. , 6.1,
       6.2, 6.3, 6.4, 6.5, 6.6, 6.7, 6.8, 6.9, 7. , 7.1, 7.2, 7.3, 7.4,
       7.5, 7.6, 7.7, 7.8, 7.9, 8. , 8.1, 8.2, 8.3, 8.4, 8.5, 8.6, 8.7,
       8.8, 8.9, 9. , 9.1, 9.2, 9.3, 9.4, 9.5, 9.6, 9.7, 9.8, 9.9])
```

In [736]:
```
#Input Variable:
# tfinal = final time (in seconds)
# Output Variables:
# t = array of times at which speed is % computed (in seconds)
# v = array of speeds (meters/second)

g = 9.81  # Acceleration in SI units
tfinal = int(input('Enter final time (in seconds): '))
dt = tfinal/500
t =np.arange(0,tfinal,dt)   # Creates an array of 501 time values
# the final velocity
v = g*t
```

```
# The Distance travel by the object
D = - (0.5*g*t**2)
plt.plot(t,v, 'g')
plt.xlabel('time in sec')
plt.ylabel('velocity in m/sec ')
plt.title('Final velocity Curve with respect to Time')
plt.show()
plt.plot(t,D, 'r')
plt.xlabel('time in sec')
plt.ylabel('Distance in m ')
plt.title('Distance with respect to Time')
plt.show()
```

```
Enter final time (in seconds): 5
```





Projectile Motion without making Custom Functions

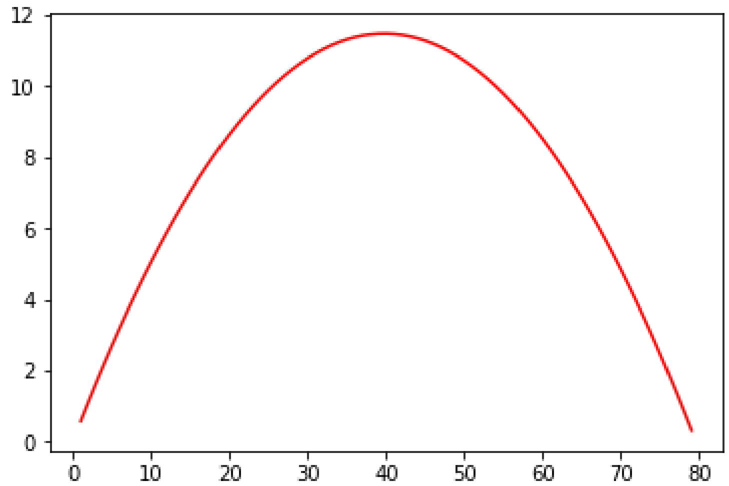
The equation of Projectile's trajectory is

$$y = \tan\theta \cdot x - \frac{g}{2 \cdot u^2 \cdot \cos^2\theta} \cdot x^2$$

In [569]:
```
# Projectile's trajectory
import math
x = np.arange(1,80, 1)
g = 9.8
```

```
v0 = 30
theta = math.radians(30)
# The equation of Projectile's trajectory is :

y = x* math.tan(theta)-(x**2 * g)/(2 * v0**2 * (math.cos(theta)**2))
plt.plot(x,y,'r')
```

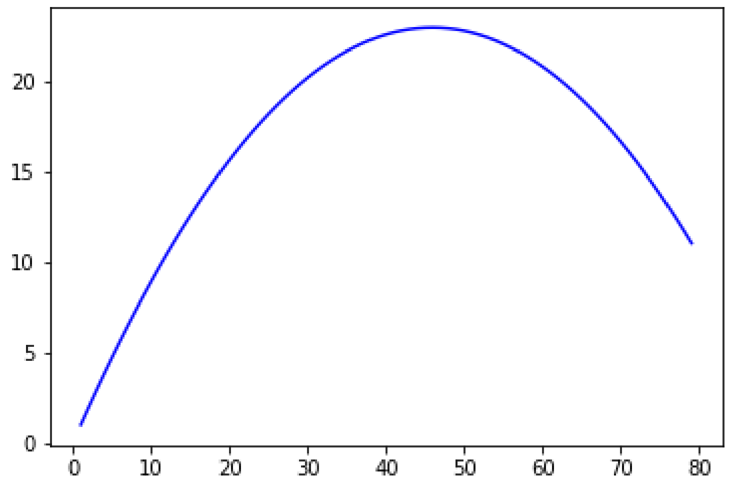
Out[569]: [<matplotlib.lines.Line2D at 0x253ccc27630>]



In [570]:

```
theta1 = math.radians(45)
y1 = x* math.tan(theta1)-(x**2 * g)/(2 * v0**2 * (math.cos(theta1)**2))
plt.plot(x,y1,'b')
```

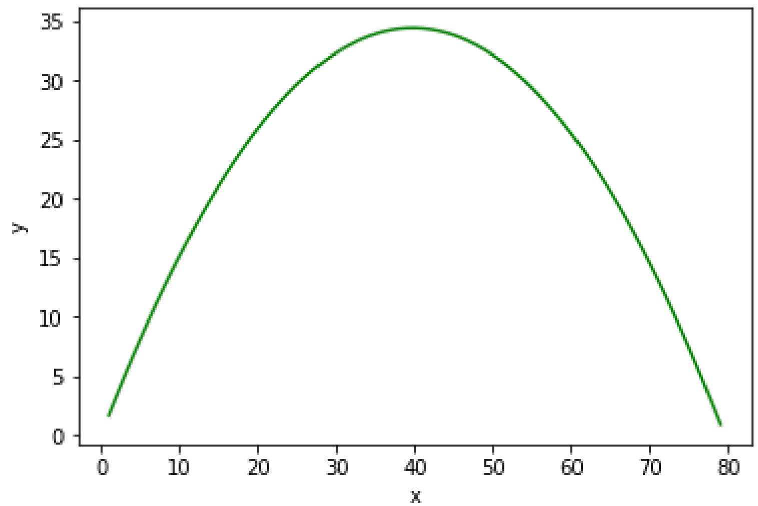
Out[570]: [<matplotlib.lines.Line2D at 0x253ccbd5a90>]



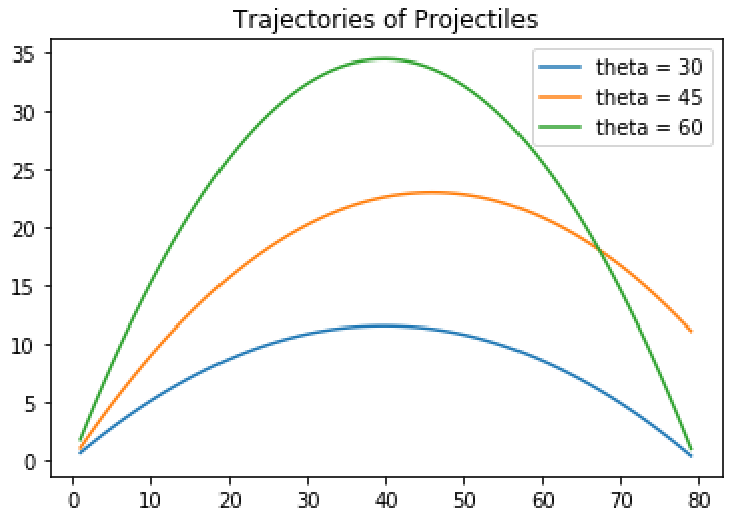
In [572]:

```
theta2 =math.radians(60)
y2 = x * math.tan(theta2)-(x**2 * g)/(2 * v0**2 * (math.cos(theta2)**2
))
plt.plot(x,y2,'g')
plt.xlabel('x')
plt.ylabel('y')
```

Out[572]: Text(0,0.5,'y')



In [573]:
```
ax = plt.subplot(111)

ax.plot(x, y, label='theta = 30')
ax.plot(x, y1, label='theta = 45')
ax.plot(x, y2, label='theta = 60')
plt.title('Trajectories of Projectiles')
ax.legend()
plt.show()
```

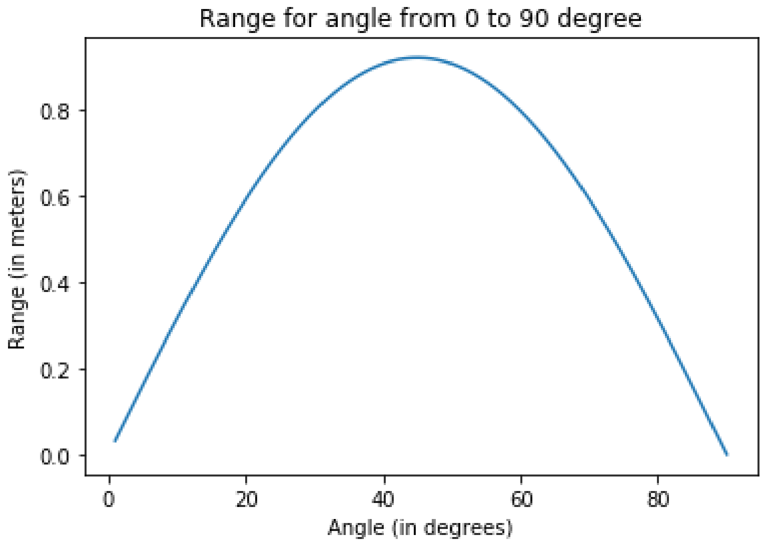


Range and Height of projectile

In [574]:
```
# Range of projectile
import math
def Range(angle):
    v = 3
    R = v**2 *(np.sin(np.radians(2*angle)))/ g
    return R

angle = np.arange(1, 90, 0.01)
```

```
plt.plot(angle, Range(angle))
plt.xlabel('Angle (in degrees)')
plt.ylabel('Range (in meters)')
plt.title ('Range for angle from 0 to 90 degree')
```

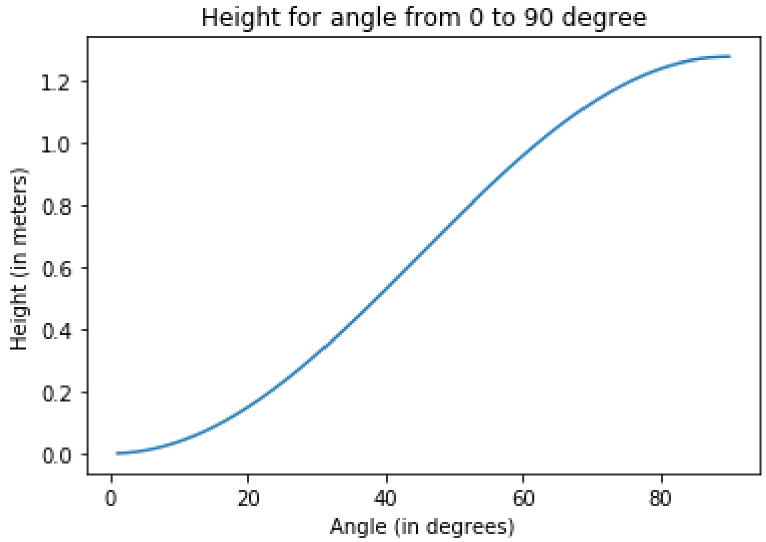
Out[574]: Text(0.5,1,'Range for angle from 0 to 90 degree')



In [575]:

```
# Height of projectile
def Height(angle):
    v = 5
    return v**2 *(np.sin(np.radians(angle))**2)/ (g*2)

angle = np.arange(1, 90, 0.01)
plt.plot(angle,Height(angle))
plt.xlabel('Angle (in degrees)')
plt.ylabel('Height (in meters)')
plt.title ('Height for angle from 0 to 90 degree')
```

Out[575]: Text(0.5,1,'Height for angle from 0 to 90 degree')



Projectile Motion by making Custom Functions

In [576]:
```
# Range of projectile
import math
def Range(angle,in_vel):
    R = in_vel**2 *(np.sin(np.radians(2*angle)))/ g
    return R

# Height of projectile
def Height(angle,in_vel):
    return in_vel**2 *(np.sin(np.radians(angle))**2)/ (g*2)

Height(45,3), Range(45,3)
```

Out[576]: $(0.22959183673469393, \quad 0.9183673469387754)$

Waves and Oscillations

Simple Harmonic Motion

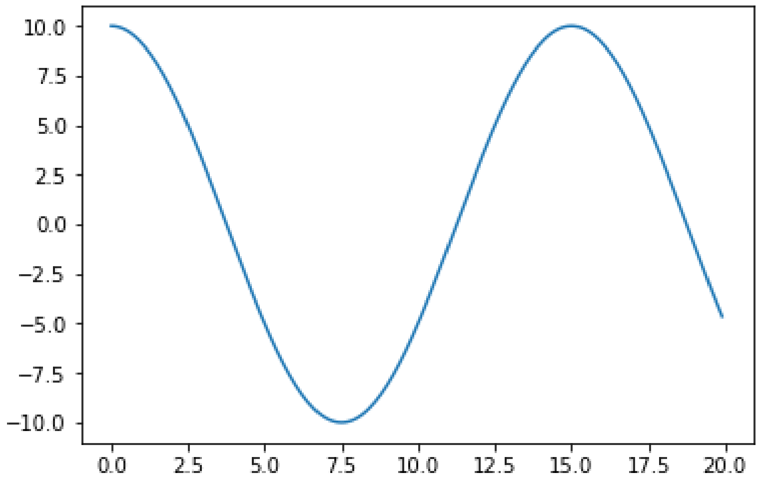
Displacement :

$x = x_m Cos(\omega t + \theta)$

- x_m is the amplitude (maximum displacement of the system)
- t is the time
- w is the angular frequency, and
- θ is the phase constant or phase

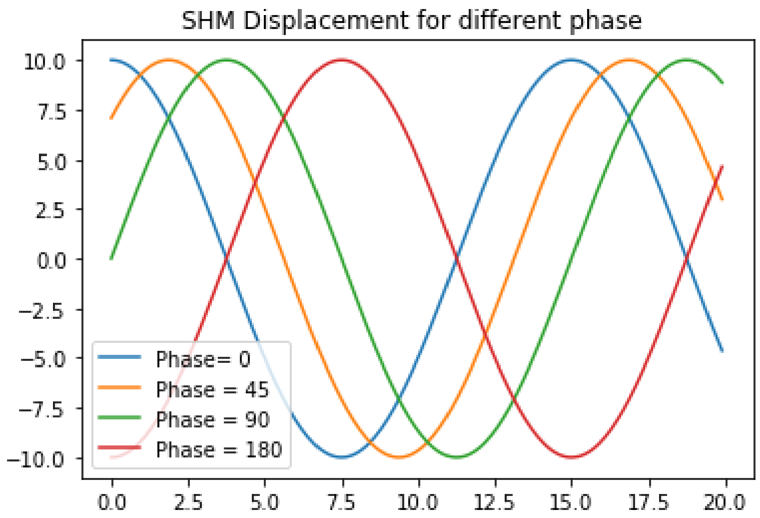
In [577]:
```
xm = 10
w = 24
phase = 0
t= np.arange(0,20, 0.1)
x = xm*np.cos(np.radians(w*t- phase))
plt.plot(t,x)
```

Out[577]:
```
[<matplotlib.lines.Line2D at 0x253ccaa3fd0>]
```



In [578]:
```
xm = 10
```

```
w = 24
phase = 0
t= np.arange(0,20, 0.1)
x = xm*np.cos(np.radians(w*t- phase))

phase1 = 45
x1 = xm*np.cos(np.radians(w*t- phase1))

phase2 = 90
x2 = xm*np.cos(np.radians(w*t- phase2))

phase3 = 180
x3 = xm*np.cos(np.radians(w*t- phase3))

ax = plt.subplot(111)
ax.plot(t,x, label='Phase= 0')
ax.plot(t,x1, label='Phase = 45')
ax.plot(t,x2, label='Phase = 90')
ax.plot(t,x3, label='Phase = 180')
plt.title('SHM Displacement for different phase ')
ax.legend()
plt.show()
```



Velocity :

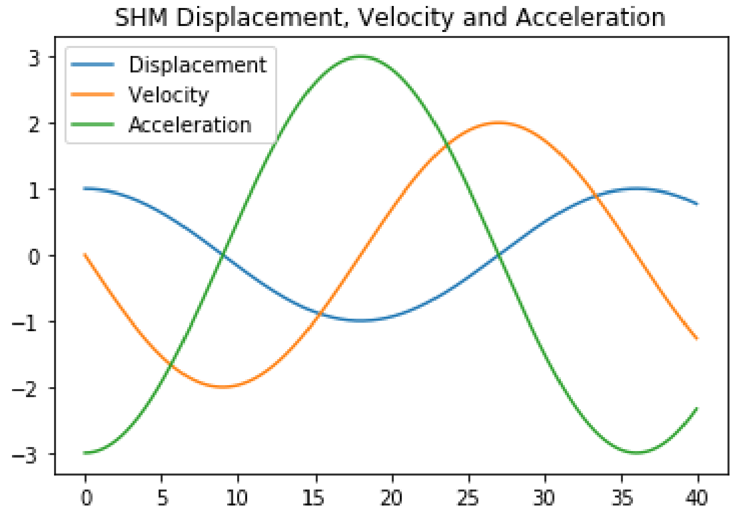
$$V = -x_m \omega Sin(\omega t + \theta)$$

Acceleration :

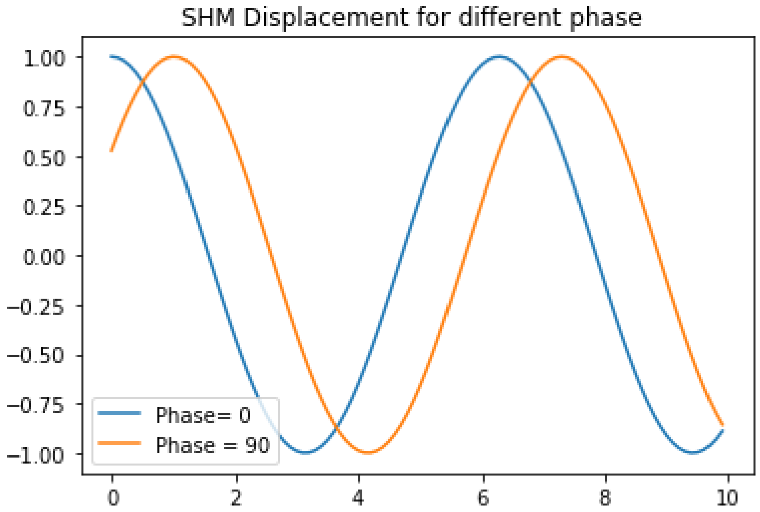
$$a = -x_m \omega^2 Sin(\omega t + \theta)$$

In [579]:

```
xm = 1
w = 10
vm= 2
am =3
phase = 0
t= np.arange(0,40, 0.1)
x = xm*np.cos(np.radians(w*t- phase))
v =- vm* np.sin(np.radians(w*t- phase))
a =- am* np.cos(np.radians(w*t- phase))
#v =- xm*w* np.sin(np.radians(w*t- phase))
#a =- xm*w**2* np.cos(np.radians(w*t- phase))

ax = plt.subplot(111)
ax.plot(t,x, label='Displacement')
ax.plot(t,v, label='Velocity ')
ax.plot(t,a, label='Acceleration')

plt.title('SHM Displacement, Velocity and Acceleration')
ax.legend()
plt.show()
```

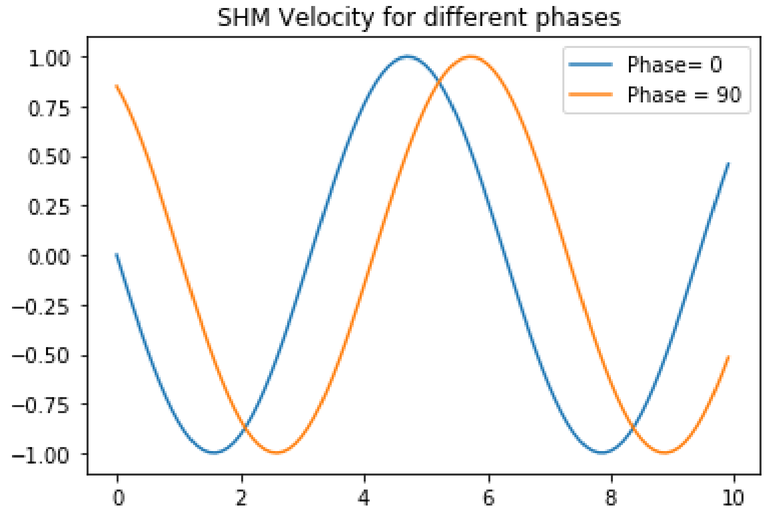


In [580]:
```
xm = 1
w = 1
phase = 0
t= np.arange(0,10, 0.1)
x1 = xm*(np.cos((w*t)- phase))
phas1 = np.pi/2
x2 = xm*(np.cos((w*t)- phase1))
#print (x2)

ax = plt.subplot(111)
ax.plot(t,x1, label='Phase= 0')
ax.plot(t,x2, label='Phase = 90')

plt.title('SHM Displacement for different phase ')
ax.legend()
plt.show()
```



In [581]:

```
vm = 1
w = 1
phase = 0
t= np.arange(0,10, 0.1)
v1 =- vm*(np.sin((w*t)- phase))
phas1 = np.pi/2
v2 =- vm*(np.sin((w*t)- phase1))
#print (x2)

ax = plt.subplot(111)
ax.plot(t,v1, label='Phase= 0')
ax.plot(t,v2, label='Phase = 90')

plt.title('SHM Velocity for different phases ')
ax.legend()
plt.show()
```



am = 1 w = 1 phase = 0 t= np.arange(0,10, 0.1) a1 = - am*(np.cos((w*t)- phase)) phas1 = np.pi/2 a2 =- am*(np.cos((w*t)- phase1))

print (x2)

ax = plt.subplot(111) ax.plot(t,a1, label='Phase= 0') ax.plot(t,a2, label='Phase = 90')

plt.title('SHM Acceleration for different phase ') ax.legend() plt.show()

Simple Harmonic Motion as Circular Motion

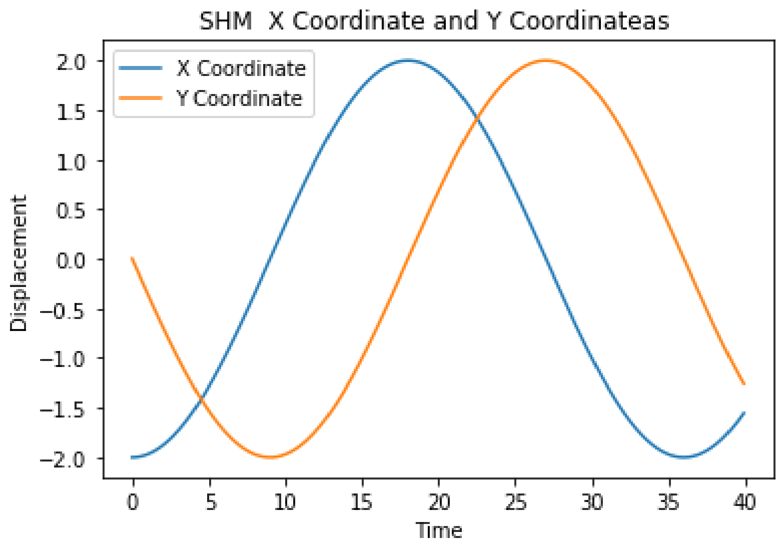
$x = x_m \omega Cos(\omega t + \theta)$

$y = y_m \omega Sin(\omega t + \theta)$

In [584]:

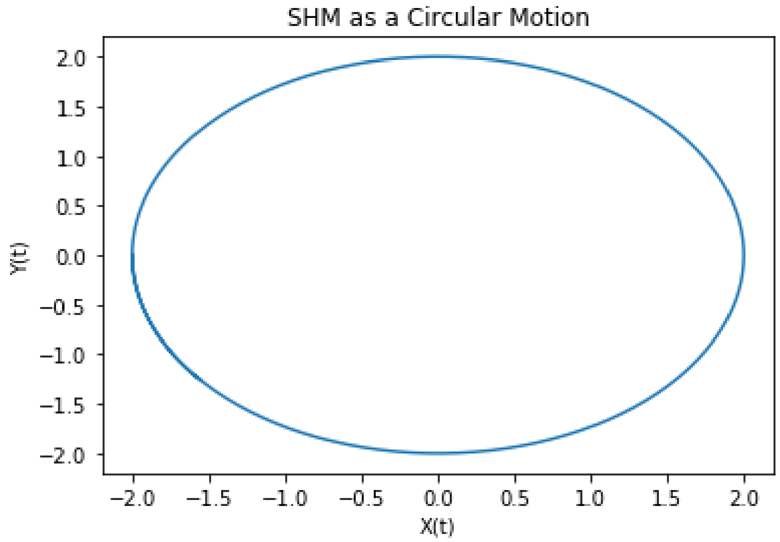
```
w = 10
xm= 2
ym =2
phase = 180
```

```
t= np.arange(0,40, 0.1)
x = xm*np.cos(np.radians(w*t- phase))
y = ym*np.sin(np.radians(w*t- phase))

ax = plt.subplot(111)
ax.plot(t,x, label='X Coordinate')
ax.plot(t,y, label='Y Coordinate')
plt.title('SHM  X Coordinate and Y Coordinateas ')
plt.xlabel('Time')
plt.ylabel('Displacement')
ax.legend()
plt.show()
```



In [585]:
```
plt.plot(x,y)
plt.xlabel('X(t)')
plt.ylabel('Y(t)')
plt.title('SHM as a Circular Motion')
```

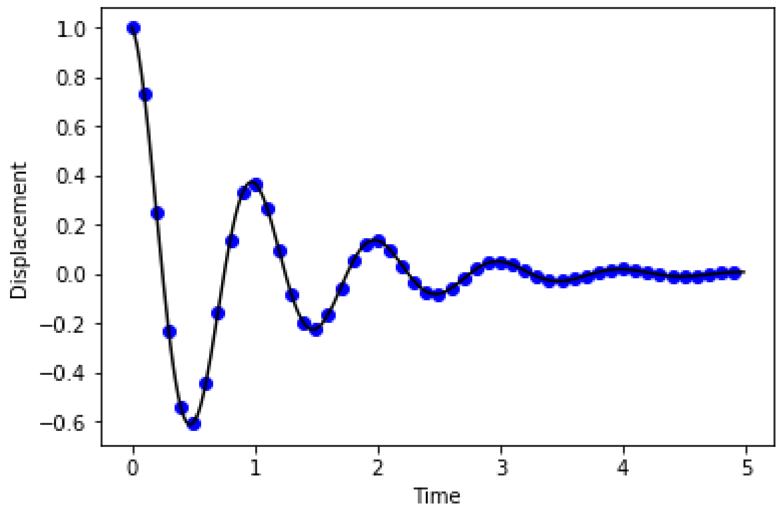
Out[585]: Text(0.5,1,'SHM as a Circular Motion')



Damped Oacilation

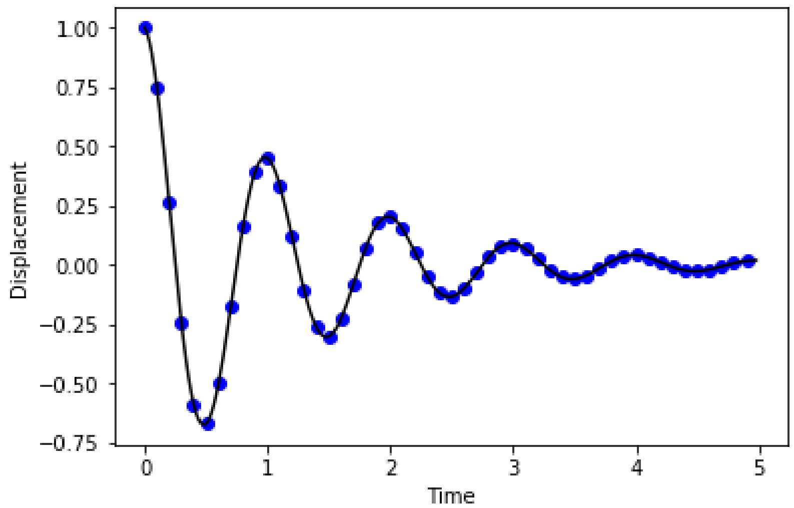
In [586]:
```
def f(t):
    return np.exp(-t) * np.cos(2*np.pi*t)

t1 = np.arange(0.0, 5.0, 0.1)
t2 = np.arange(0.0, 5.0, 0.02)

plt.figure()
plt.subplot(111)
plt.plot(t1, f(t1), 'bo', t2, f(t2), 'k')
plt.xlabel('Time')
plt.ylabel('Displacement')
```

Out[586]:
```
Text(0,0.5,'Displacement')
```



In [587]:
```
def f(t):
    b = 4
    m = 10
    w = 2*np.pi
    p=0
    return np.exp(-2*b*t/m) * np.cos(w*t - p)    # w = srtq(k/m)

t1 = np.arange(0.0, 5.0, 0.1)
t2 = np.arange(0.0, 5.0, 0.02)

plt.figure()
plt.subplot(111)
plt.plot(t1, f(t1), 'bo', t2, f(t2), 'k')
plt.xlabel('Time')
plt.ylabel('Displacement')
```

Out[587]:
```
Text(0,0.5,'Displacement')
```



In [737]:
```
b = 6
m = 5
w = np.pi*2


if int(b) == int(2*m*w):
    print (" This Critical Under Damped condition:",int(b),'=', int(2*m
*w ))

elif int(b) <= int(2*m*w):
    print (" This is Under Damped condition:", int(b),'<',int(2*m*w) )

elif int(b) >= int(2*m*w):
    print (" This is Over Dapmed condition: ", int(b),'>', int(2*m*w) )
```

```
 This is Under Damped condition: 6 < 62
```

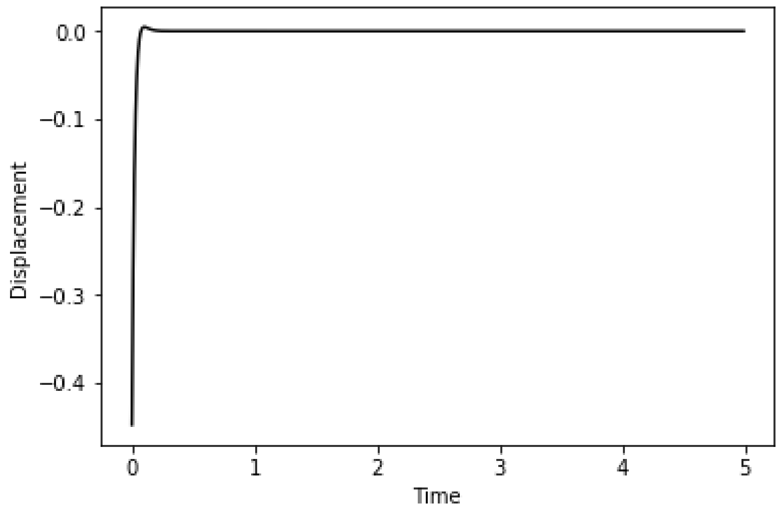
Under Damped , Critical Damped and Over Dapmed

In [740]:
```
def f(t,b,m,w,p):
```

```
    if int(b) == int(2*m*w):
        print (" This Critical Under Damped condition:",int(b),'=', int
(2*m*w ))

    elif int(b) <= int(2*m*w):
        print (" This is Under Damped condition:", int(b),'<',int(2*m*w
) )

    elif int(b) >= int(2*m*w):
        print (" This is Over Dapmed condition: ", int(b),'>', int(2*m*
w) )

    x = np.exp(-2*b*t/m) * (np.cos(w*t - p) )

    return x    # w = srtq(k/m)

t1 = np.arange(0.0, 5.0, 0.01)
t2 = np.arange(0.0, 5.0, 0.02)


plt.figure()
plt.subplot(111)
plt.plot(t1, f(t1,89,5,2*np.pi,90), 'k')
plt.xlabel('Time')
plt.ylabel('Displacement')
```

```
 This is Over Dapmed condition:  89 > 62
```

Out[740]:
```
Text(0,0.5,'Displacement')
```



Electricity and Magnetism

Coulomb Force between Charges

Coulomb law for two point charges:

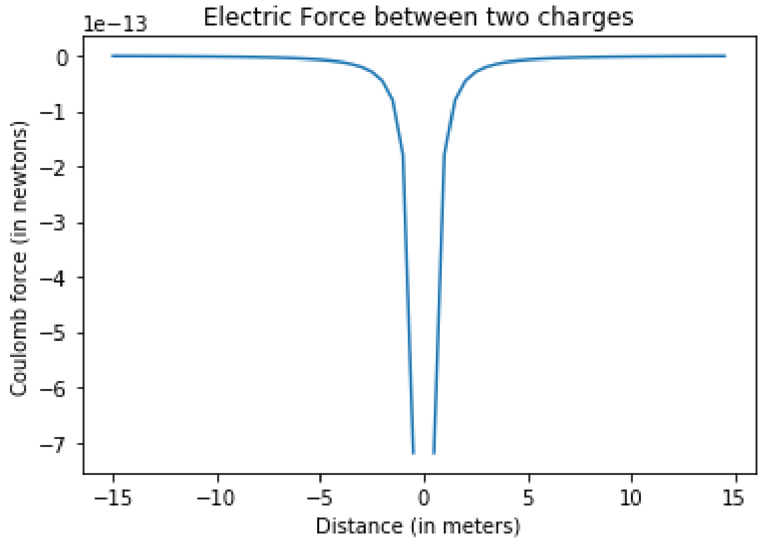
$$F = \frac{kq_1q_2}{r^2}$$

In [593]:
```
ep = 8.854* 10**-12
k = 1/(4*np.pi*ep)
q1 = -1* 10**-13
q2 = +2* 10**-10
r = np.arange(-15,15, 0.5)

F = (k*q1*q2)/(r**2)
```

```
plt.plot(r,F)
plt.xlabel('Distance (in meters)')
plt.ylabel('Coulomb force (in newtons)')
plt.title('Electric Force between two charges')
```

```
D:\anaconda\lib\site-packages\ipykernel_launcher.py:7: RuntimeWarning:
divide by zero encountered in true_divide
  import sys
```

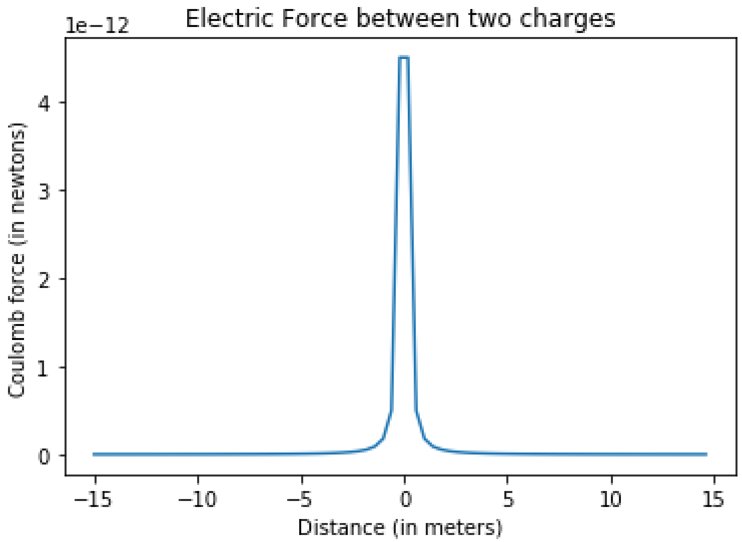
Out[593]: Text(0.5,1,'Electric Force between two charges')



In [594]:
```
ep = 8.854* 10**-12
k = 1/(4*np.pi*ep)
q1 = +1* 10**-13
q2 = +2* 10**-10
r = np.arange(-15,15, 0.4)

F = (k*q1*q2)/(r**2)

plt.plot(r,F)
plt.xlabel('Distance (in meters)')
```

```
plt.ylabel('Coulomb force (in newtons)')
plt.title('Electric Force between two charges')
```

Out[594]: Text(0.5,1,'Electric Force between two charges')



Grivatational Force

This program calculates and displays the Gravitational Force between two masses:

$$F = \frac{Gm^1m^2}{r^2}$$

In [596]:

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

# define function of gravitational force
def grv_force(r):
    G = 6.67*10**-11
    gf = (G*2*4)/(r**2)
    return gf
r = np.arange(1, 10, 0.1)
```

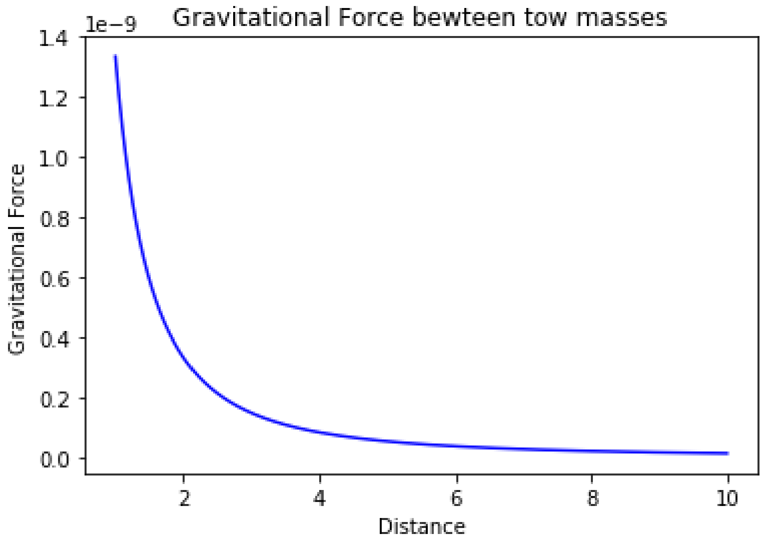
```
plt.plot(r, grv_force(r), 'g')
plt.xlabel('Distance')
plt.ylabel('Gravitational Force')
plt.title('Gravitational Force bewteen tow masses')
plt.show()
print (r,grv_force(r))
```


```
[1.  1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2.  2.1 2.2 2.3 2.4 2.5 2.6 2.
7
 2.8 2.9 3.  3.1 3.2 3.3 3.4 3.5 3.6 3.7 3.8 3.9 4.  4.1 4.2 4.3 4.4 4.
5
 4.6 4.7 4.8 4.9 5.  5.1 5.2 5.3 5.4 5.5 5.6 5.7 5.8 5.9 6.  6.1 6.2 6.
3
 6.4 6.5 6.6 6.7 6.8 6.9 7.  7.1 7.2 7.3 7.4 7.5 7.6 7.7 7.8 7.9 8.  8.
1
 8.2 8.3 8.4 8.5 8.6 8.7 8.8 8.9 9.  9.1 9.2 9.3 9.4 9.5 9.6 9.7 9.8 9.
9] [5.33600000e-10 4.40991736e-10 3.70555556e-10 3.15739645e-10
 2.72244898e-10 2.37155556e-10 2.08437500e-10 1.84636678e-10
 1.64691358e-10 1.47811634e-10 1.33400000e-10 1.20997732e-10
 1.10247934e-10 1.00869565e-10 9.26388889e-11 8.53760000e-11
 7.89349112e-11 7.31961591e-11 6.80612245e-11 6.34482759e-11
 5.92888889e-11 5.55254943e-11 5.21093750e-11 4.89990817e-11

 4.61591696e-11 4.35591837e-11 4.11728395e-11 3.89773557e-11
```

```
 4.61591696e-11 4.35591837e-11 4.11728395e-11 3.89773557e-11
 3.69529086e-11 3.50821828e-11 3.33500000e-11 3.17430101e-11
 3.02494331e-11 2.88588426e-11 2.75619835e-11 2.63506173e-11
 2.52173913e-11 2.41557266e-11 2.31597222e-11 2.22240733e-11
 2.13440000e-11 2.05151865e-11 1.97337278e-11 1.89960840e-11
 1.82990398e-11 1.76396694e-11 1.70153061e-11 1.64235149e-11
 1.58620690e-11 1.53289285e-11 1.48222222e-11 1.43402311e-11
 1.38813736e-11 1.34441925e-11 1.30273437e-11 1.26295858e-11
 1.22497704e-11 1.18868345e-11 1.15397924e-11 1.12077295e-11
 1.08897959e-11 1.05852013e-11 1.02932099e-11 1.00131357e-11
 9.74433893e-12 9.48622222e-12 9.23822715e-12 8.99983134e-12
 8.77054569e-12 8.54991187e-12 8.33750000e-12 8.13290657e-12
 7.93575253e-12 7.74568152e-12 7.56235828e-12 7.38546713e-12
 7.21471065e-12 7.04980843e-12 6.89049587e-12 6.73652317e-12
 6.58765432e-12 6.44366622e-12 6.30434783e-12 6.16949936e-12
 6.03893164e-12 5.91246537e-12 5.78993056e-12 5.67116590e-12
 5.55601833e-12 5.44434241e-12]
```

In [597]:

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

# define function of gravitational force
def grv_force(r,m1,m2):
    G = 6.67*10**-11
    gf = (G*m1*m2)/(r**2)
    return gf
r = np.arange(1, 10, 0.01)
plt.plot(r, grv_force(r,4,5), 'b')
plt.xlabel('Distance')
plt.ylabel('Gravitational Force')
plt.title('Gravitational Force bewteen tow masses')
plt.show()
#print (r,grv_force(r))
```


Gravitational Force bewteen tow masses
1e−9
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Gravitational Force
2
4
6
8
10
Distance

In [ ]: