MAT451:Python Programming for Mathematical Modelling- E-Record

Arnav Sinha, 2040811

Class: 4 EMS

Topic 1: Recapitulation of Python Commands and Programming Structure

Date:- 4/1/22

Aim:

To go through basic python commands, syntax and other basic concepts. Specifically concepts like list, arrays, matrices etc is to be revised in this topic.

Code:

In [1]:
```
num1=int(input("Enter a number: "))
num2=int(input("Enter second number: "))
add=num1+num2
print(add)
```

```
Enter a number: 4
Enter second number: 6
10
```

In [6]:
```
sub=num1-num2
sub
```

Out[6]:
```
-2
```

In [4]:
```
mul=num1*num2
mul
```

Out[4]:
```
24
```

In [5]:
```
div=num1/num2
div
```

Out[5]:
```
0.6666666666666666
```

In [ ]:
```
## Use of if, elif and else functions
```

In [2]:
```
num=int(input("enter a number: "))
if num%2==0:
    print("number is even")
else:
    print("number is odd")
```

```
enter a number: 5
number is odd
```

In [9]:
```
number=int(input("enter a number: "))
if number>0:
    print("number is positive")
elif number==0:
    print("number is zero")
else:
    print("number is negative")
```

```
enter a number: -10
number is negative
```

In [2]:
```
a=int(input("enter first number (a): "))
b=int(input("enter second number (b): "))
c=int(input("enter third number (c): "))
if a>b and a>c:
    print("a is largest number")
elif c>b and c>b:
    print("c is largest number")
elif b>a and b>c:
    print("b is largest number")
```

```
enter first number (a): 5
enter second number (b): 3
enter third number (c): 12
c is largest number
```

In [1]:
```
f=1;
n=int(input("enter number: "))
while(n>0):
    f=f*n
    n=n-1
print(f)
```

```
enter number: 7
5040
```

In [2]:
```
num=int(input("enter no: "))
i=1;
while(i<11):
    print(num," x ",i," = ",num*i)
    i=i+1
```

```
enter no: 4
4  x  1  =  4
4  x  2  =  8
4  x  3  =  12
4  x  4  =  16
4  x  5  =  20
4  x  6  =  24
4  x  7  =  28
4  x  8  =  32
4  x  9  =  36
4  x  10  =  40
```

List

In [7]:
```
list=[]

num=int(input("enter no of elements: "))
while (num>0):
    m=int(input("enter no to be added: "))
    list.append(m)
    num=num-1
print("Revised list: ")
print(list)
```

```
enter no of elements: 5
enter no to be added: 1
enter no to be added: 2
enter no to be added: 3
enter no to be added: 4
enter no to be added: 5
Revised list:
[1, 2, 3, 4, 5]
```

In [8]:
```
print("First element in the list: ",list[0])
```

```
First element in the list:  1
```

In [9]:
```
list.remove(4)
print("Updated list: ",list)
```

```
Updated list:  [1, 2, 3, 5]
```

In [10]:
```
list1=[]

num=int(input("enter no of elements: "))
while (num>0):
    m=int(input("enter no to be added: "))
    list1.append(m)
    num=num-1
print("List: ")
print(list1)

print("First element in the list: ",list1[0])
print("Second element in the list: ",list1[1])
print("Third element in the list: ",list1[2])
print("Fourth element in the list: ",list1[3])


list1.remove(4)
print("Updated list: ",list1)
```

```
enter no of elements: 4
enter no to be added: 1
enter no to be added: 2
enter no to be added: 3
enter no to be added: 4
List:
[1, 2, 3, 4]
First element in the list:  1
Second element in the list:  2
Third element in the list:  3
Fourth element in the list:  4
Updated list:  [1, 2, 3]
```

Arrays

In [12]:
```
from numpy import *
array=array([[1,2,3],[4,5,6],[7,8,9]],dtype=float)
print(array)
```

```
[[1. 2. 3.]
 [4. 5. 6.]
 [7. 8. 9.]]
```

In [13]:
```
print(zeros(3))
```

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

In [14]:
```
print(ones(5))
```

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

In [15]:
```
print(identity(4))
```

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

In [16]:
```
print(arange(2,30,4))
## Starting number, ending number and spaces in between
```

```
[ 2  6 10 14 18 22 26]
```

In [17]:
```
print(linspace(2,10,5))
## Starting number, ending number and number of points in between
```

```
[ 2.  4.  6.  8. 10.]
```

Matrices

In [17]:
```
from numpy import *
matrix1=array([[1,2,3],[4,5,6],[7,8,9]])
print(matrix1)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

In [20]:
```
## 2nd row
matrix1[1]
```

Out[20]:
```
array([4, 5, 6])
```

In [22]:
```
## 1st column
matrix1[:,0]
```

Out[22]:
```
array([1, 4, 7])
```

In [23]:
```
## Obtaining elements
matrix1[1,2]
```

Out[23]:
```
6
```

In [3]:
```
print(sin(matrix1))
```

```
[[ 0.84147098  0.90929743  0.14112001]
 [-0.7568025  -0.95892427 -0.2794155 ]
 [ 0.6569866   0.98935825  0.41211849]]
```

In [7]:
```
## Programme to add matrices

matrix1=array([[1,2,3],[4,5,6],[7,8,9]])
matrix2=array([[1,2,5],[4,5,9],[4,8,9]])
print("m1=\n",matrix1)
print("m2=\n",matrix2)
matrix3=matrix1+matrix2
print("The addition of the matrices results in")
print(matrix3)
```

```
m1=
 [[1 2 3]
 [4 5 6]
 [7 8 9]]
m2=
 [[1 2 5]
 [4 5 9]
 [4 8 9]]
The addition of the matrices results in
[[ 2  4  8]
 [ 8 10 15]
 [11 16 18]]
```

In [8]:
```
## Programme to subtract matrices

matrix1=array([[1,2,3],[4,5,6],[7,8,9]])
matrix2=array([[1,2,5],[4,5,9],[4,8,9]])
print("m1=\n",matrix1)
print("m2=\n",matrix2)
matrix3=matrix1-matrix2
print("The subtraction of the matrices results in")
print(matrix3)
```

```
m1=
 [[1 2 3]
 [4 5 6]
 [7 8 9]]
m2=
 [[1 2 5]
 [4 5 9]
 [4 8 9]]
The subtraction of the matrices results in
[[ 0  0 -2]
 [ 0  0 -3]
 [ 3  0  0]]
```

In [11]:
```
## Dot product

print("The dot product of the matrices is: ")
matrix1*matrix2
```

```
The dot product of the matrices is:

array([[ 1,  4, 15],
       [16, 25, 54],
       [28, 64, 81]])
```

In [12]:
```
## Quotient: Division of each element

print("The division of the matrices gives: ")
matrix1/matrix2
```

```
The division of the matrices gives:

array([[1.        , 1.        , 0.6       ],
       [1.        , 1.        , 0.66666667],
       [1.75      , 1.        , 1.        ]])
```

Programme to take matrix input from user in Python

In [13]:
```
R = int(input("Enter the number of rows:"))
C = int(input("Enter the number of columns:"))

# Initialize matrix
matrix = []
print("Enter the entries rowwise:")

# For user input
for i in range(R):          # A for loop for row entries
    a =[]
    for j in range(C):      # A for loop for column entries
         a.append(int(input()))
    matrix.append(a)

# For printing the matrix
for i in range(R):
    for j in range(C):
        print(matrix[i][j], end = " ")
    print()
```

```
Enter the number of rows:4
Enter the number of columns:4
Enter the entries rowwise:
1
2
3
4
5
6
7
8
9
0
1
2
3
4
5
6
1 2 3 4
5 6 7 8
9 0 1 2
3 4 5 6
```

In [18]:
```
mat=matrix(input("Enter a matrix"))
print(mat)
```

```
Enter a matrix1 2; 3 4
[[1 2]
 [3 4]]
```

In [7]:
```
## Multiplication of matrix

from numpy import *
A = array([[1,3,5],[7,9,11],[13,15,17]])
B = array([[2,4,6],[8,10,12],[14,16,18]])
np.dot(A,B)
```

Out[7]:
```
array([[ 96, 114, 132],
       [240, 294, 348],
       [384, 474, 564]])
```

In [4]:
```
## Using for loop

A = [[1,2,3],
    [4,5,6],
    [7,8,9]]
B = [[11,32,33],
    [41,25,61],
    [71,38,19]]

result = [] # final result
for i in range(len(A)):

    row = [] # the new row in new matrix
    for j in range(len(B[0])):

        product = 0 # the new element in the new row
        for v in range(len(A[i])):
            product += A[i][v] * B[v][j]
        row.append(product) # append sum of product into the new row

    result.append(row) # append the new row into the final result
print(result)
```

```
[[306, 196, 212], [675, 481, 551], [1044, 766, 890]]
```

Conclusion:

We successfully learnt the basic commands in python. Namely, topics like lists, matrices, arrays and other basic functions were learnt and examples for each were given. Other basic concepts of python programming were also learnt.

Topic 2- 2D and 3D plotting and Graph Customization

Date:- 18/1/22

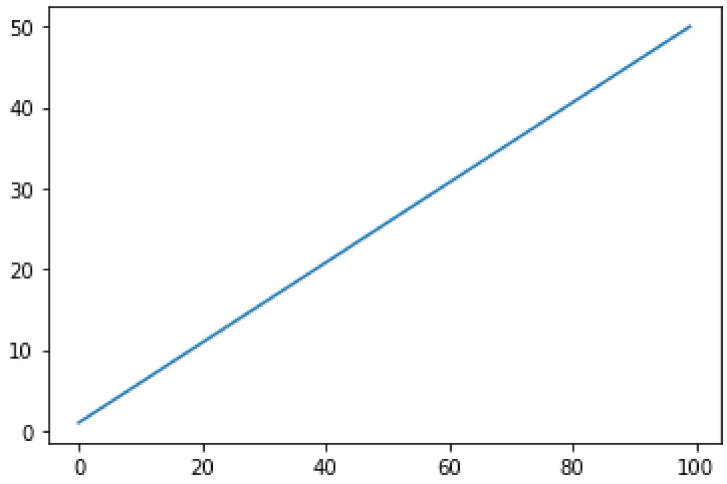
Aim:

To use python to generate 2D and 3D graphs of various functions. This also includes plots like scatter plots, surface plots etc.

Code:

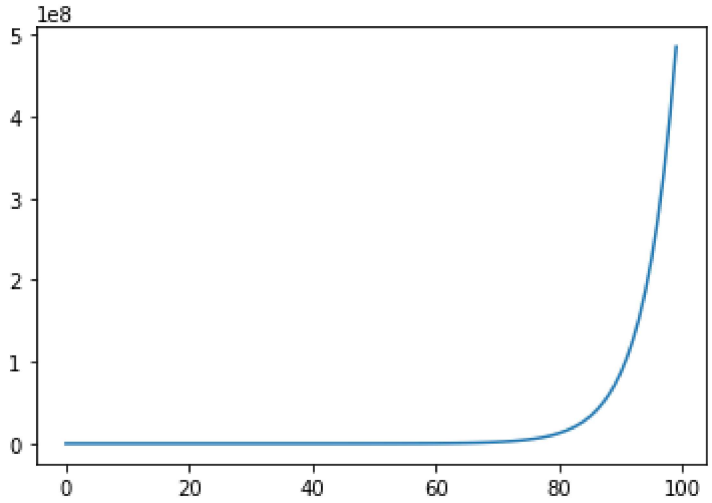
In [2]:
```
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
x=np.linspace(1,50,100)
plt.plot(x)
plt.show()
```

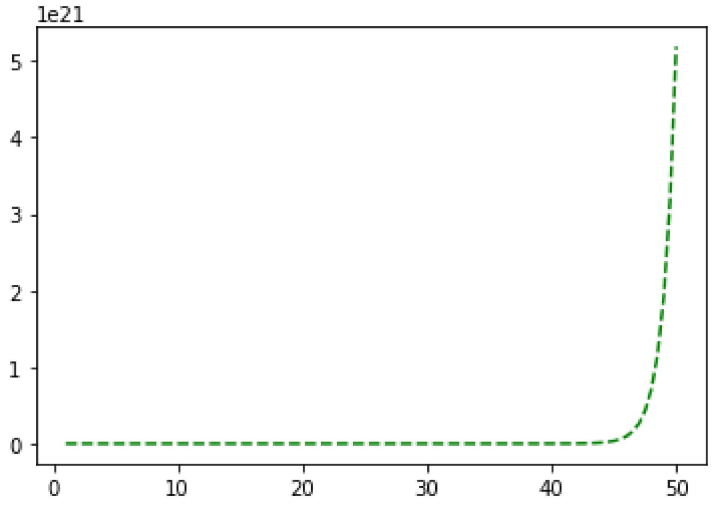


Write programme codes to draw graph of an exponential function.

In [3]:
```
X= np.linspace(1,20,100)
plt.plot(np.exp(X))
plt.show()
```

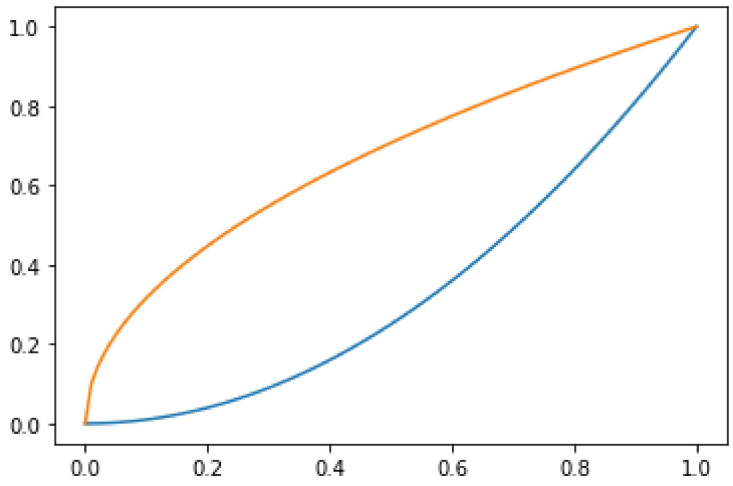


In [4]:
```
X = np.linspace(1,50,100)
plt.plot(X,np.exp(X),'--',color='green')
plt.show()
```



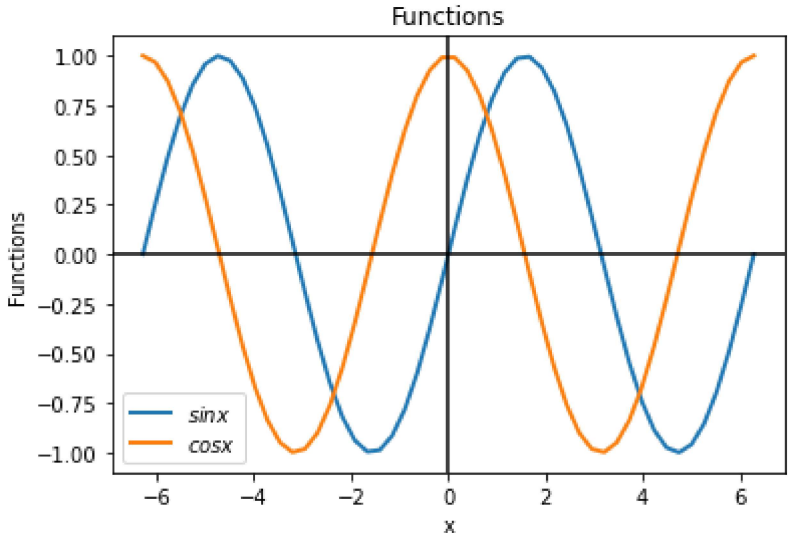
Write a python programme to plot x^2 and sqrt(x) in same plot

In [9]:
```
X = np.linspace(0,1,100)
y=X**2
z=(X)**(0.5)
plt.plot(X,y)
plt.plot(X,z)
plt.show()
```



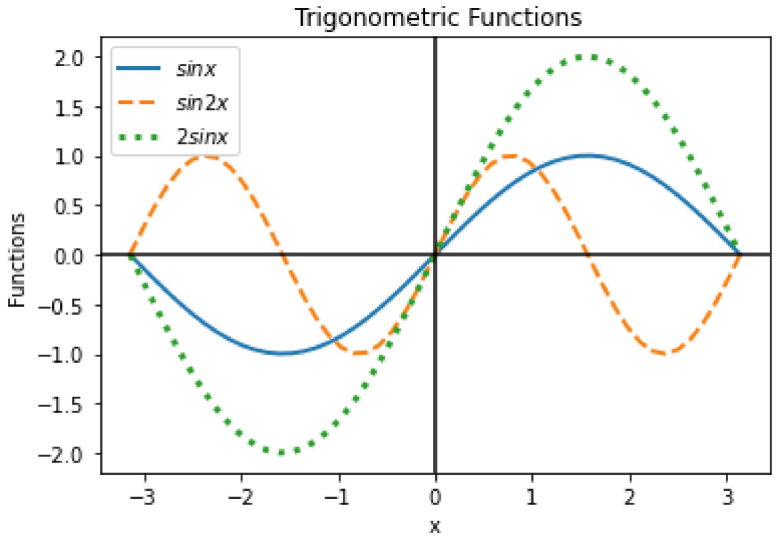
Plot sinx and cosx in the same plot.

In [24]:
```
X = np.linspace(-2*np.pi,2*np.pi)
y=np.sin(X)
z=np.cos(X)
plt.plot(X,y,lw=2,label='$sinx$')
plt.plot(X,z,lw=2,label='$cosx$')

plt.title("Functions")
plt.xlabel("x")
plt.ylabel("Functions")

plt.axhline(0,color='black')
plt.axvline(0,color='black')
plt.legend()
None
```



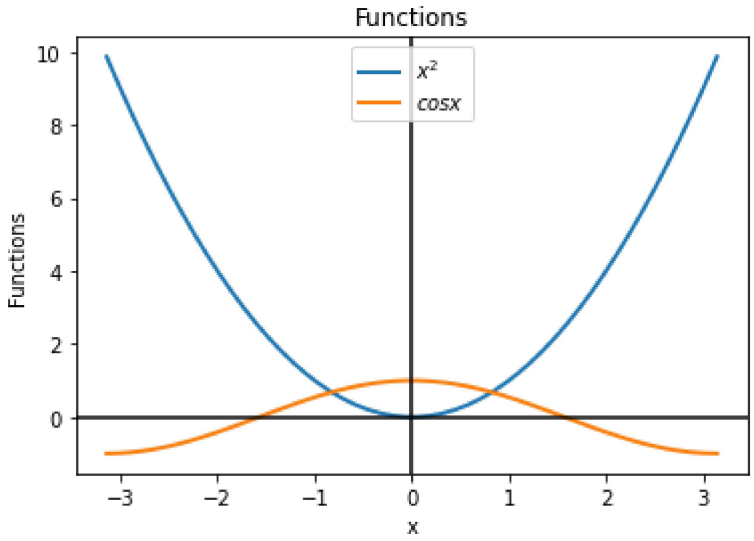
Plot sinx, sin2x and 2sinx in a single plot.

In [9]:
```
X = np.linspace(-np.pi,np.pi)
x=np.sin(X)
y=np.sin(2*X)
z=2*np.sin(X)
plt.plot(X,x,lw=2,label='$sinx$')
plt.plot(X,y,lw=2,label='$sin2x$',linestyle='dashed')
plt.plot(X,z,lw=3,label='$2sinx$',linestyle='dotted')

plt.title(" Trigonometric Functions")
plt.xlabel("x")
plt.ylabel("Functions")

plt.axhline(0,color='black')
plt.axvline(0,color='black')
plt.legend()
None
```



Plot x^2 and cosx in the same plot.

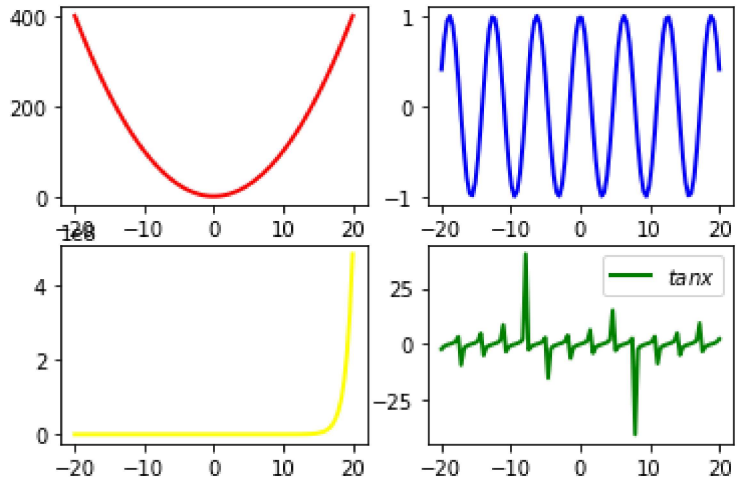
In [21]:
```
X = np.linspace(-np.pi,np.pi)
y=X**2
z=np.cos(X)
plt.plot(X,y,lw=2,label='$x^2$')
plt.plot(X,z,lw=2,label='$cosx$')

plt.title("Functions")
plt.xlabel("x")
plt.ylabel("Functions")

plt.axhline(0,color='black')
plt.axvline(0,color='black')
plt.legend()
None
```



Plot subplots: i)e^x, ii) x^2 , iii) cosx , iv) tanx

In [23]:
```
x=np.linspace(-20,20,100)
z=np.square(x)
a=np.cos(x)
y=np.exp(x)
b=np.tan(x)

plt.title('line plot')
plt.subplot(2,2,1)
plt.plot(x,z, lw=2, color='red', label='$x^2$')
plt.subplot(2,2,2)
plt.plot(x,a, lw=2, color='blue', label='$cosx$')
plt.subplot(2,2,3)
plt.plot(x,y, lw=2, color='yellow', label='$e^x$')
plt.subplot(2,2,4)
plt.plot(x,b, lw=2, color='green', label='$tanx$')

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

```
None
```



Given are the heights and weights of students in a class. Plot a line chart, using the following specifications.

In [1]:
```
height = [121.9,124.5,129.5,134.6,139.7,147.3,152.4, 157.5,162.6]
weight= [19.7,21.3,23.5,25.9,28.5,32.1,35.7,39.6,43.2]
```

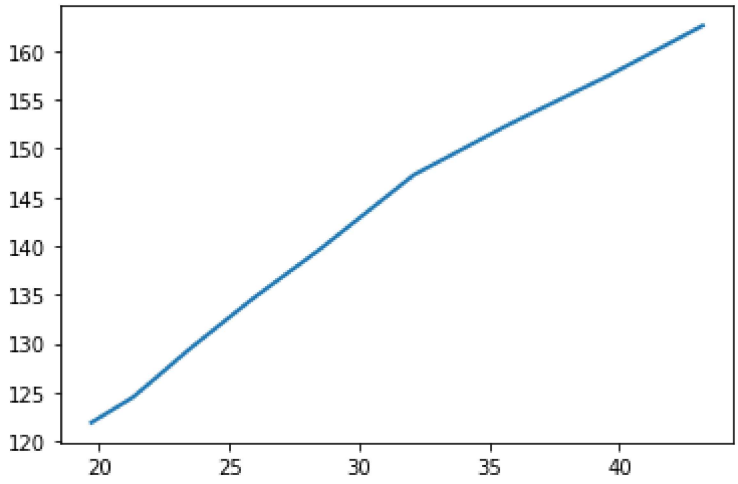
In [2]:
```
height
```

Out[2]:
```
[121.9, 124.5, 129.5, 134.6, 139.7, 147.3, 152.4, 157.5, 162.6]
```

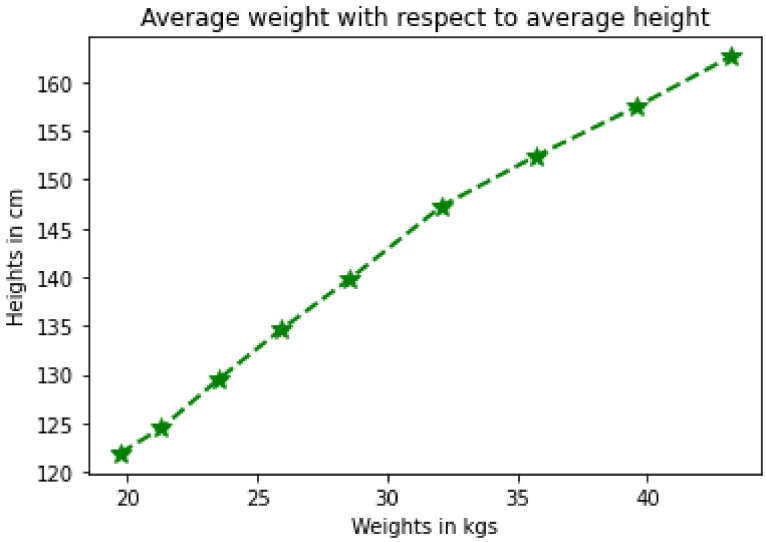
In [3]:
```
weight
```

Out[3]:
```
[19.7, 21.3, 23.5, 25.9, 28.5, 32.1, 35.7, 39.6, 43.2]
```

In [7]:
```
## x axis represents weight and y axis represents height

plt.plot(weight,height,lw=2)
plt.show()
```



In [13]:
```
## Label x axis, y axis, give title, give color, line width and marker to the sam

plt.title("Average weight with respect to average height")
plt.xlabel("Weights in kgs")
plt.ylabel("Heights in cm")
plt.plot(weight,height,lw=2,linestyle='dashed',color="green",marker="*",markersiz
plt.show()
```

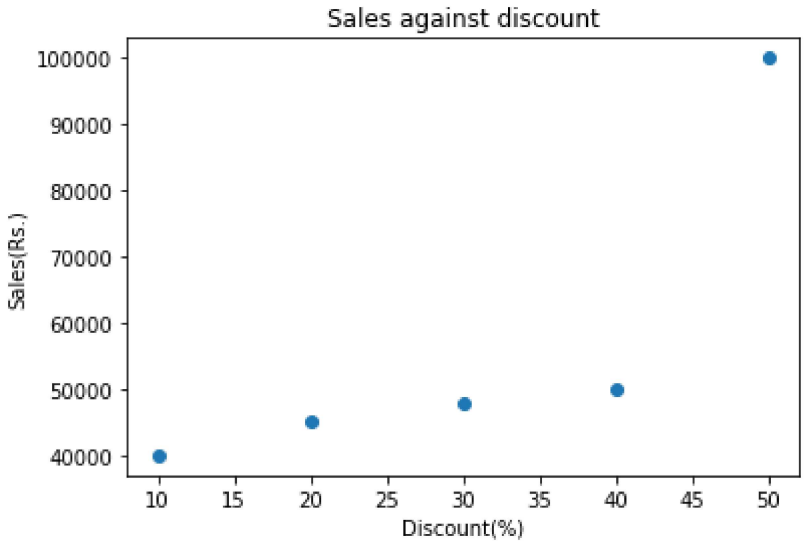


Scatter plot

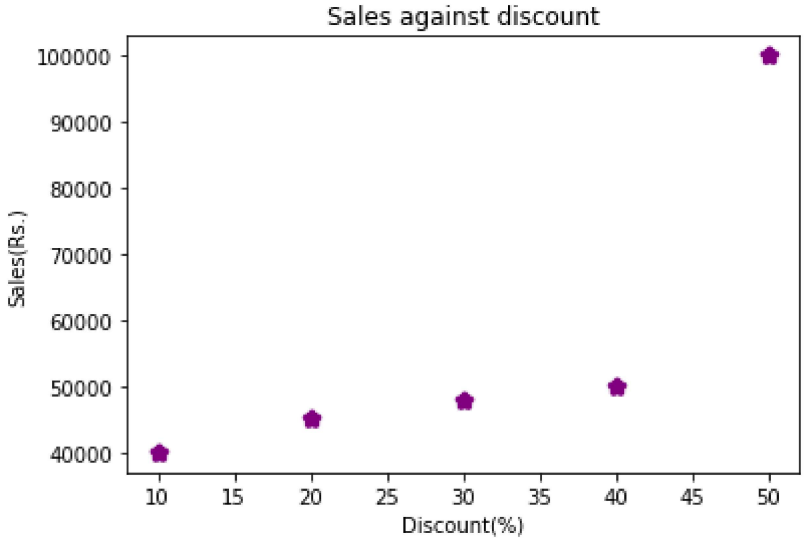
Scatter plot is a set of points plotted on horizontal and vertical axes. It is used to study correlation between two variables

Q) A shop which sells designer Kurtas, gave discounts ranging from 10% to 50% over a period of 5 weeks, during the sales season. They made a sales of 40000,45000,48000,50000,100000 (in Rs) respectively. They recorded sales for each type of discount in an array. Draw a scatter plot to show a relationship between the discount offered and sales made.

In [15]:
```
sales=[40000,45000,48000,50000,100000]
disc=[10,20,30,40,50]

plt.title("Sales against discount")
plt.xlabel("Discount(%)")
plt.ylabel("Sales(Rs.)")
plt.scatter(disc,sales)
plt.show()
```



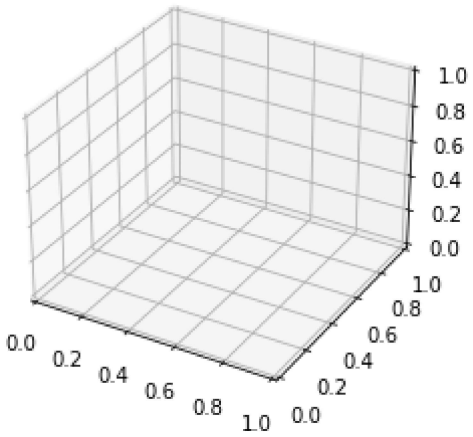
In [17]:
```
## Modifying aesthetics
plt.title("Sales against discount")
plt.xlabel("Discount(%)")
plt.ylabel("Sales(Rs.)")
plt.scatter(disc,sales,lw=4,marker='*',color='purple')
plt.show()
```



3D plots

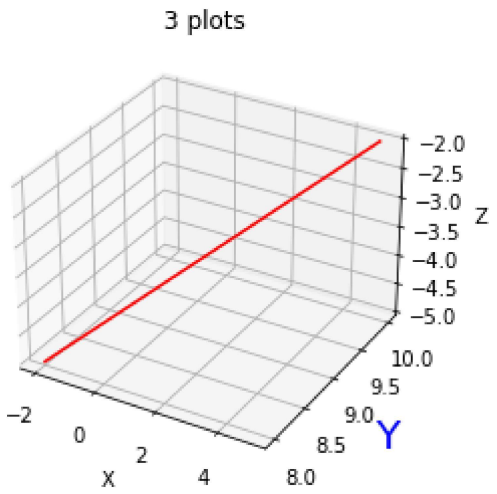
In [11]:
```
from mpl_toolkits import mplot3d
import numpy as np
from numpy import *
import matplotlib.pyplot as plt
```

In [3]:
```
plt.axes(projection="3d")
None
```



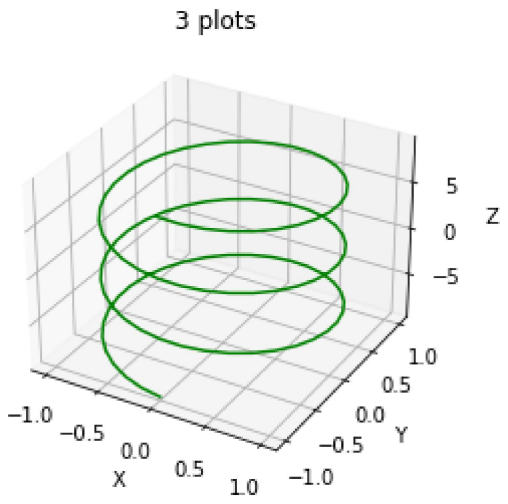
Draw a 3d plot for x=-2+7t, y=8+2t, z=-5+3t

In [8]:
```
ax= plt.axes(projection="3d")

t=linspace(0,1,10)
x=-2+7*t
y=8+2*t
z=-5+3*t

ax.plot3D(x,y,z,'red')

ax.set_title("3 plots")
ax.set_xlabel("X")
ax.set_ylabel("Y",size=20, color='blue')
ax.set_zlabel("Z",fontsize=10)
None
```



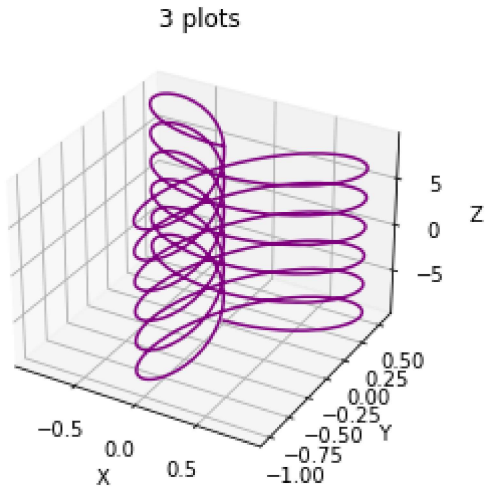
Plot r(t)=(sint)i+(cost)j+tk

In [24]:
```
ax= plt.axes(projection="3d")

t=linspace(-3*np.pi,3*np.pi,100)
x=np.sin(t)
y=np.cos(t)
z=t

ax.plot3D(x,y,z,'green')

ax.set_title("3 plots")
ax.set_xlabel("X")
ax.set_ylabel("Y")
ax.set_zlabel("Z")
None
```



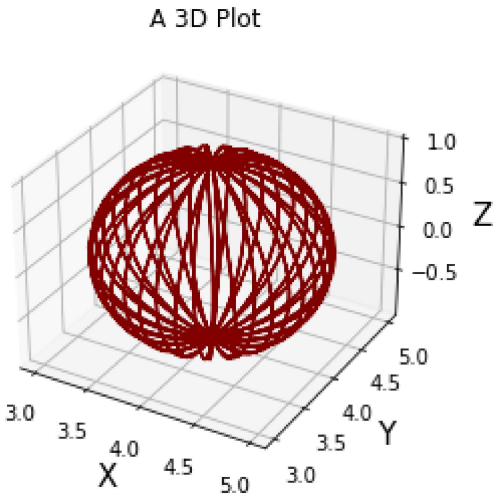
Plot r(t)=(sin3t)(cost)i+(sin3t)(sint)j+tk

In [23]:
```
ax= plt.axes(projection="3d")

t=linspace(-3*np.pi,3*np.pi,1000)
##more the points in between (like 1000), smoother the curve
x=np.sin(3*t)*np.cos(t)
y=np.sin(3*t)*np.sin(t)
z=t

ax.plot3D(x,y,z,'purple')

ax.set_title("3 plots")
ax.set_xlabel("X")
ax.set_ylabel("Y")
ax.set_zlabel("Z")
None
```



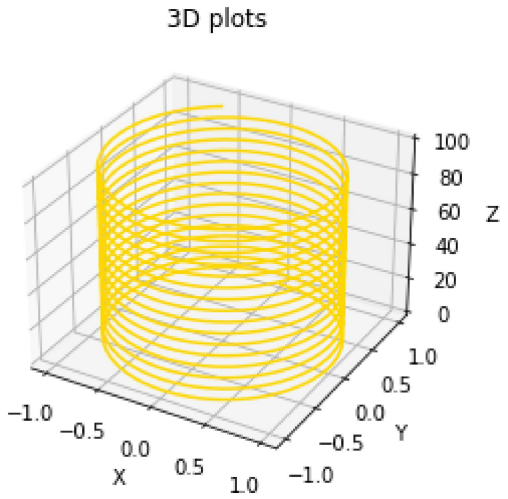
Plot r(t)=[4+(sin20t)(cost)]i+[4+(sin20t)(sint)]j+cos(20t)k

In [22]:
```
import numpy as np
ax=plt.axes(projection="3d")

t=linspace(-2*np.pi,2*np.pi,1000)
xval=4+np.sin(20*t)*np.cos(t)
yval=4+np.sin(20*t)*np.sin(t)
zval=np.cos(20*t)
ax.plot3D(xval,yval,zval,'maroon')

plt.title("A 3D Plot")

ax.set_xlabel("X",fontsize=15)
ax.set_ylabel("Y",fontsize=15)
ax.set_zlabel("Z",fontsize=15)
None
```

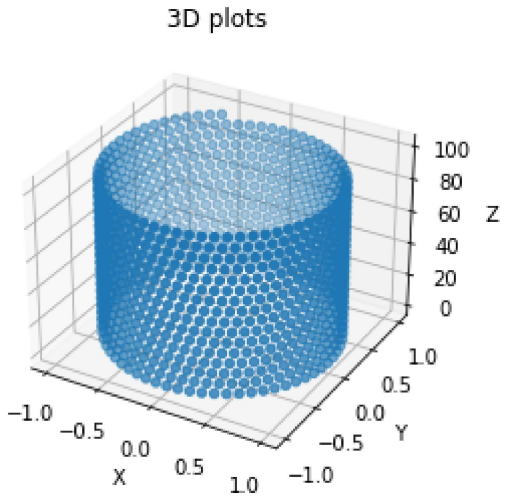


Plot x= sin(z), y = cos(z)

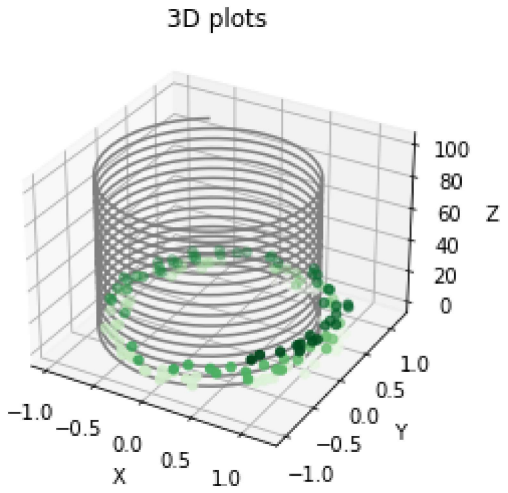
In [30]:
```
## Method 1

ax= plt.axes(projection="3d")

z=linspace(0,100,1000)
x=np.sin(z)
y=np.cos(z)

ax.plot3D(x,y,z,'gold')

ax.set_title("3D plots")
ax.set_xlabel("X")
ax.set_ylabel("Y")
ax.set_zlabel("Z")
None
```



In [32]:
```
##Method 2: using scatter plot

ax= plt.axes(projection="3d")

z=linspace(1,100,1000)
x=np.sin(z)
y=np.cos(z)

ax.scatter3D(x,y,z)

ax.set_title("3D plots")
ax.set_xlabel("X")
ax.set_ylabel("Y")
ax.set_zlabel("Z")
None
```



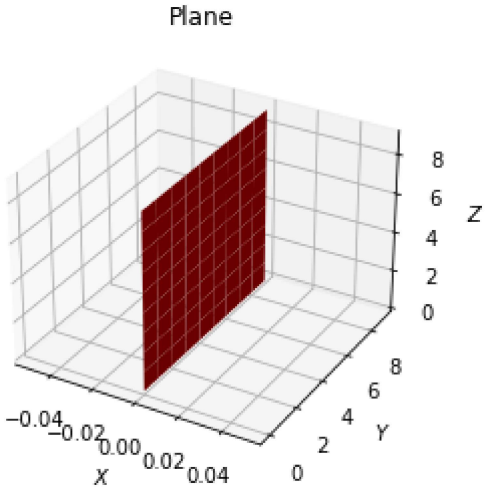
In [35]:
```
## Method 3

ax= plt.axes(projection="3d")

z=linspace(1,100,1000)
x=np.sin(z)
y=np.cos(z)
ax.plot3D(x,y,z,color='gray')

Z=15*random.random(150)
X=np.sin(Z)+0.2*random.random(150)
Y=np.cos(Z)+0.2*random.random(150)

ax.scatter3D(X,Y,Z,c=Z,cmap='Greens')

ax.set_title("3D plots")
ax.set_xlabel("X")
ax.set_ylabel("Y")
ax.set_zlabel("Z")
None
```



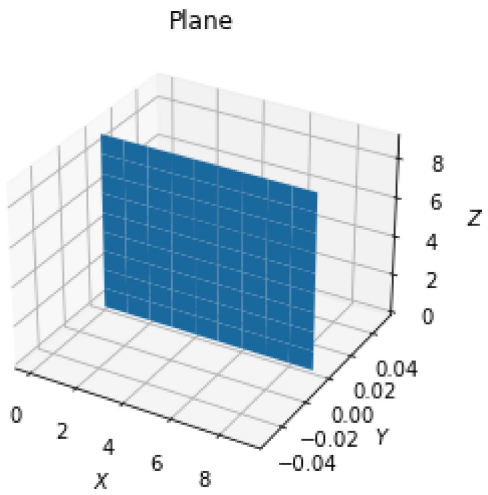
Surface plots

In [5]:
```
from numpy import *
import matplotlib.pyplot as plt
from matplotlib.pyplot import *
from mpl_toolkits.mplot3d import Axes3D
```

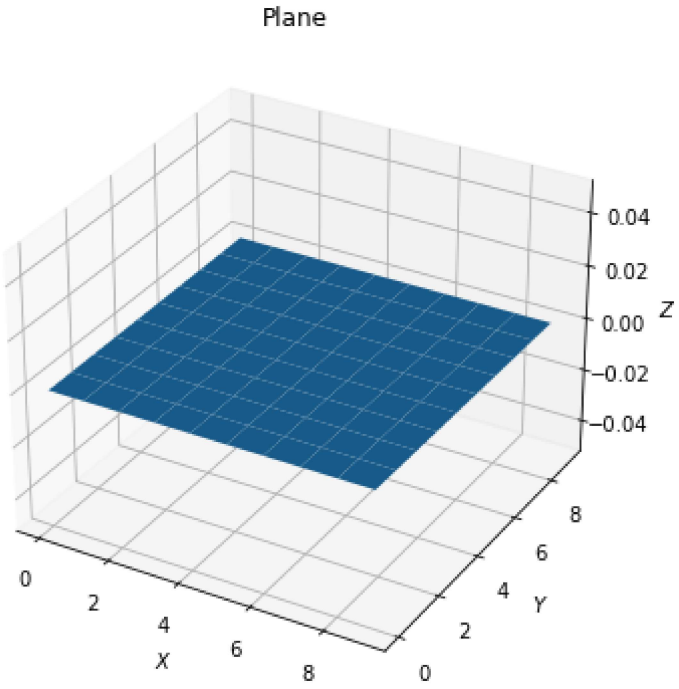
In [9]:
```
## For yz plane

yy,zz=meshgrid(range(10),range(10))

x=0

ax=figure().gca(projection='3d')
ax.plot_surface(x,yy,zz,color='red')

title("Plane")
ax.set_xlabel('$X$')
ax.set_ylabel('$Y$')
ax.set_zlabel('$Z$')
show()
```



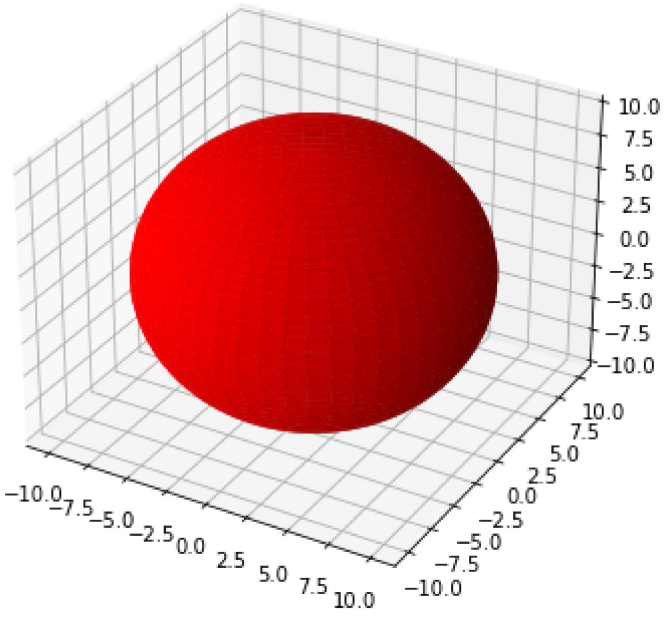
In [6]:
```
## For xz plane

xx,zz=meshgrid(range(10),range(10))

y=0

ax=figure().gca(projection='3d')
ax.plot_surface(xx,y,zz)

title("Plane")
ax.set_xlabel('$X$')
ax.set_ylabel('$Y$')
ax.set_zlabel('$Z$')
show()
```



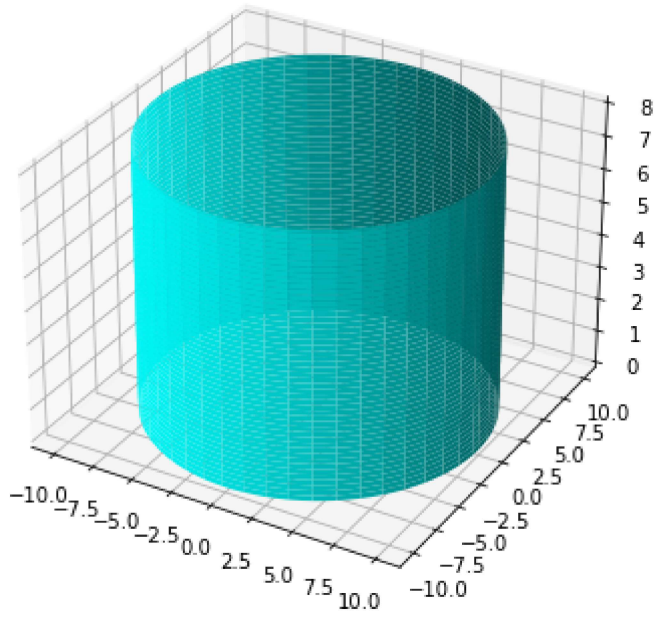
In [16]:
```
## For xy plane

xx,yy=meshgrid(range(10),range(10))
z=0*xx-0*yy-0

fig=plt.figure(figsize=(8,6))
az=fig.add_subplot(1,1,1,projection='3d')
az.plot_surface(xx,yy,z)

title("Plane")
az.set_xlabel('$X$')
az.set_ylabel('$Y$')
az.set_zlabel('$Z$')
show()
```



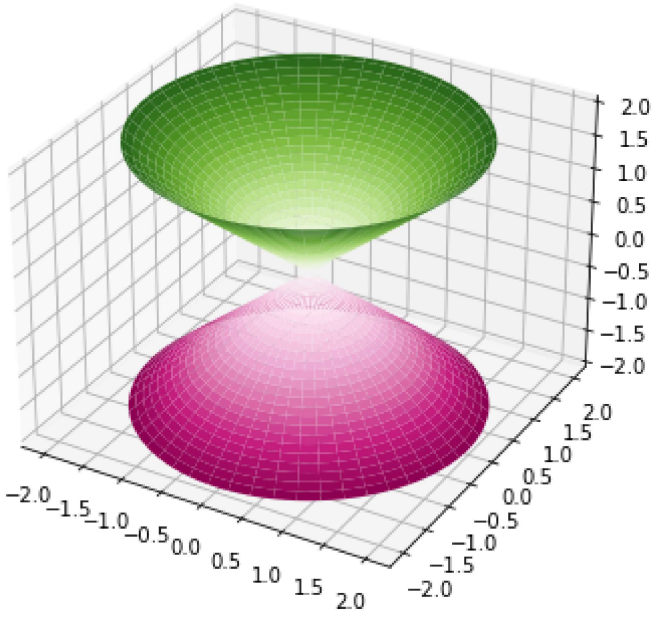
In [9]:
```
fig=plt.figure(figsize=(8,6))
ax=fig.add_subplot(1,1,1,projection='3d')

u=linspace(0,2*pi,100)
v=linspace(0,pi,100)
x=10*outer(cos(u),sin(v))
y=10*outer(sin(u),sin(v))
z=10*outer(ones(size(v)),cos(v))

ax.plot_surface(x,y,z,color='r')
show()
```



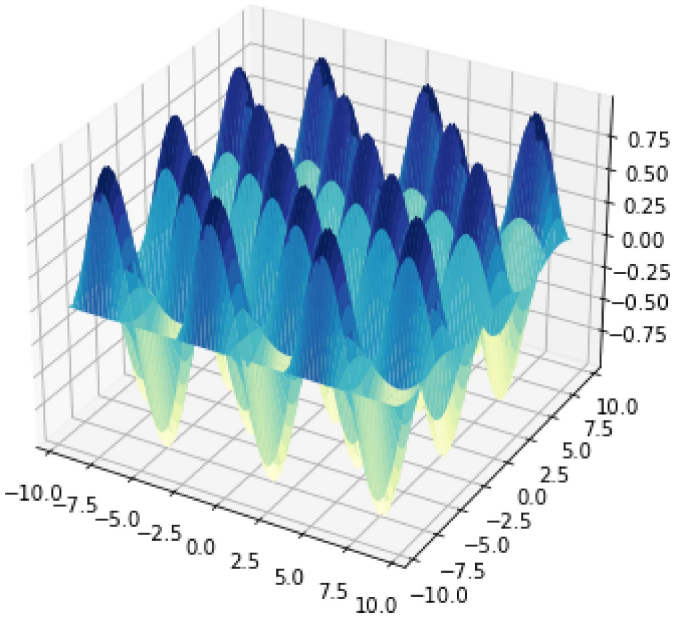
In [11]:
```
fig=plt.figure(figsize=(8,6))
ax=fig.add_subplot(1,1,1,projection='3d')

u=linspace(0,2*pi,100)
v=linspace(0,8,100)

x=10*outer(ones(size(u)),cos(u))
y=10*outer(ones(size(u)),sin(u))
z=outer(v,ones(size(v)))

ax.plot_surface(x,y,z,color='cyan')
show()
```



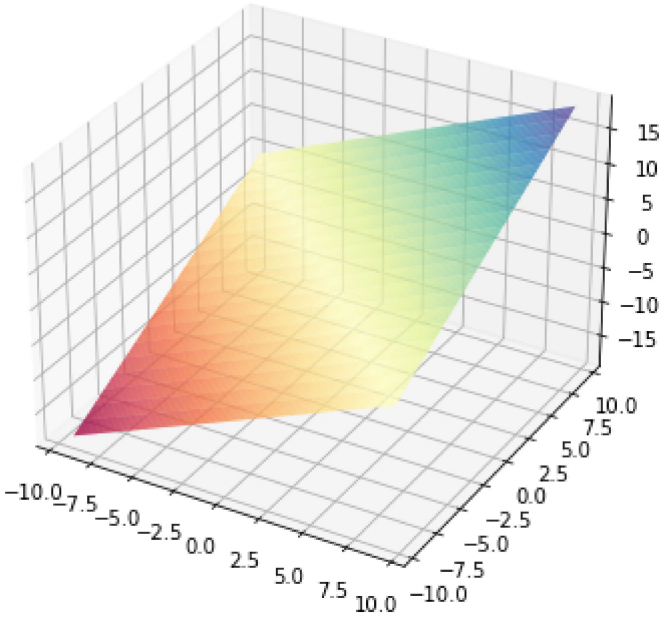
In [12]:
```
fig=plt.figure(figsize=(8,6))
ax=fig.add_subplot(1,1,1,projection='3d')

u=linspace(0,2*pi,100)
v=linspace(-2,2,100)

x=outer(v,cos(u))
y=outer(v,sin(u))
z=outer(v,ones(size(u)))

ax.plot_surface(x,y,z,cmap='PiYG')
show()
```



In [13]:
```
fig=plt.figure(figsize=(8,6))
ax=fig.add_subplot(1,1,1,projection='3d')

x=arange(-3*pi,3*pi,0.1)
y=arange(-3*pi,3*pi,0.1)

xx,yy=meshgrid(x,y)
z=sin(xx)*sin(yy)

ax.plot_surface(xx,yy,z,cmap='YlGnBu',cstride=2)
show()
```



In [14]:
```
fig=plt.figure(figsize=(8,6))
ax=fig.add_subplot(1,1,1,projection='3d')

x=arange(-3*pi,3*pi,0.1)
y=arange(-3*pi,3*pi,0.1)

xx,yy=meshgrid(x,y)
z=xx+yy

ax.plot_surface(xx,yy,z,cmap=cm.Spectral,cstride=1)
show()
```



Conclusion:

2D and 3D plots were visualised. Complex functions were ploltted. Subplots, surface plots, scatter plots etc were also plotted.

Topic 3: Solving calculus problems: functions, limits using python.

Date - 1/2/22

Aim:

To use python to define functions, solve basic problems regardng functions as well as complex caculus problems like solving limits of functions. Further, in the aim of this section is to use the many inbuilt math functions to solve various problems.

Functions used:

We use several functions from Python's standard library's maths module, such as:

math.acos() returns the arc cosine of the number

math.acosh() returns inverse hyperbolic cosine of number

math.asin() returns the arc sine of the number

math.asinh() returns inverse hyperbolic sine of number

math.atan() returns the arc tangent of the number in radians

math.atanh() returns inverse hyperbolic tangent of number

math.atan2() returns the arc tangent of y/x in radians

math.ceil() rounds number up to the nearest integer

math.floor() rounds number down to the nearest integer

math.comb() returns number of ways to choose k items from n items without repetition and order

math.perm() returns number of ways to choose k items from n items with order and without repetition

math.cos() returns the cosine of the number

math.cosh() returns hyperbolic cosine of number

math.sin() returns the sine of the number

math.sinh() returns hyperbolic sine of number

math.sinh() returns hyperbolic sine of number
math.tan() returns the tangent of the number

math.tanh() returns hyperbolic tangent of number

math.degrees() converts radians to degrees

math.exp() returns 'e' raised to the power of x

math.expm1() returns 'e' raised to the power of x-1

math.dist() returns Euclidian distance between 2 points p and q

math.fabs() retyrns absolute value of a number

math.factorial() returns factorial of a number

math.fmod() returns remainder of x/y

math.fsum() returns sum of all items in an iterable (tupples, lists, etc)

math.gamma() returns gamma function of x

math.gcd() returns greatest common divisor of 2 integers

math.hypot() retursn Euclidean norm

math.isqrt() rounds square root of a number down to nearest integer

math.log() returns natural log of a number or the log of a number to the base

math.log10() returns log base-10 of a number

math.log1p() returns natural log of 1+x

math.log2() returns log base-2 of x

math.pow() returns x to the power of y

math.prod() returns product of all elements in an iterable

math.radians() converts degrees to radians

math.remainder() returns closest value that can make numerator completely be divisible by denominator

math.sqrt() returns square root of a number

math.trunc() returns truncated integer parts of a number

Code:

In [1]:
```
import math
```

In [2]:
```
math.sin(math.pi/2)
```

Out[2]:
```
1.0
```

In [5]:
```
x=math.pi
math.sin(x)
```

Out[5]:
```
1.2246467991473532e-16
```

In [7]:
```
x=Symbol('x')
y=sympy.sin(x)
print(y)
```

```
sin(x)
```

In [9]:
```
sympy.sin(math.pi/2)
```

Out[9]:
$$1.0$$

Write a code to check if x+5 is greater than 0

In [10]:
```
x=Symbol('x', positive=True)## positive is used to define x as a positive number
y=x+5
if y>0:
    print("x+5 is greater than 0")
else:
    print("x+5 is less than 0")
```

```
x+5 is greater than 0
```

In [11]:
```
## We use solve() function to represent one variable in terms of another
x=Symbol('x')
y=Symbol('y')
solve(2*x+3*y-5,y)
```

Out[11]:
```
[5/3 - 2*x/3]
```

Derive quadratic formula for solving quadratic polynomial ax^2+ bx+ c

In [14]:
```
x=Symbol('x')
a=Symbol('a')
b=Symbol('b')
c=Symbol('c')
solve(a*x**2+b*x+c,x)
```

Out[14]:
```
[(-b + sqrt(-4*a*c + b**2))/(2*a), -(b + sqrt(-4*a*c + b**2))/(2*a)]
```

Derive the expression for the time it takes for a body in projectile motion to reach the highest point if it's thrown with initial velocity u at an angle theta. (At highest point the vertical component of a projectile motion is 0. That is, $u\sin(\theta) - gt = 0$.)

In [23]:
```
t=Symbol('t')
g=Symbol('g')
u=Symbol('u')
r=Symbol('r')
solve(u*sin(r)-g*t,t)
```

Out[23]:
```
[u*sin(r)/g]
```

Derive the expression for determining hypotenuse of a right angled triangle (Pythagoras Theorem). Using that find the hypotenuse of a right angled triangle with given base and altitude.

In [31]:
```
a=Symbol('a')
b=Symbol('b')
c=Symbol('c')
ans=solve(c**2-a**2-b**2,c)
ans1=ans[1]
print("hypotenuse = ", ans1)

alt= float(input("Give altitude: "))
base = float(input("Give base: "))
res = ans1.subs({a:alt, b:base})
print("when altitude =", alt, "and base = ", base, "hypotenuse =", res )
```

```
hypotenuse =  sqrt(a**2 + b**2)
Give altitude: 6
Give base: 8
when altitude = 6.0 and base =  8.0 hypotenuse = 10.0000000000000
```

Lambda Functions

1. This function can have any number of arguments but only one expression, which is evaluated and returned.
2. Lambda fucntion can be used in whenever functions objects are required.
3. Lambda functions are syntactially restricted to a single expression.

In [1]:
```
from sympy import *
f = lambda x: x**2+2
f(2)
```

Out[1]: 6

In [2]:
```
y = Symbol('y')
f = lambda x: x**2+2
pprint(f(y))
```

```
 2
y  + 2
```

Addition, subtraction, multiplication, division and composition of functions¶

In [5]:
```
from sympy import *
f = lambda x: exp(x)+x**2
g = lambda x: 2*x+1
x=Symbol('x')
```

In [6]:
```
f(x)+g(x)
```

Out[6]: $x^2 + 2x + e^x + 1$

In [7]:
```
f(1)+g(5)
```

Out[7]: $e + 12$

In [8]:
```
f(x)*g(x)
```

Out[8]: $(2x + 1)\left(x^2 + e^x\right)$

In [9]:
```
f(g(x))
```

Out[9]: $(2x + 1)^2 + e^{2x+1}$

In [10]:
```
import math
def f(x):
    func_fx=(math.exp(x))+(x*x)
    return func_fx;

def g(x):
    func_gx=(2*x)+1;
    return func_gx;

print("f(1)=",f(1))
print("g(1)=",g(1))

def add(f, g):

    return lambda x: (f(x)+g(x))
```

```
f(1)= 3.718281828459045
g(1)= 3
```

In [11]:
```
from sympy import *
x = Symbol('x')
f = lambda x: sin(x)
g= lambda x: 2*x + 1
f(1)+g(x)
```

Out[11]: $2x + \sin(1) + 1$

given f(x) = 3*x and g(x) = 2x+5. verify that f(g(x)) != g(f(x))

In [12]:
```
from sympy import *
f = lambda x: 3*x
g= lambda x: 2*x + 5
print('f(g(x)) = ',f(g(x)))
print('g(f(x)) = ',g(f(x)))

if f(g(x))!= g(f(x)):
    print('f(g(x)) is not equal to g(f(x))')
else:
    print('f(g(x)) is equal to g(f(x))')
```

```
f(g(x)) =  6*x + 15
g(f(x)) =  6*x + 5
f(g(x)) is not equal to g(f(x))
```

Write a code that accepts functions from users and performs various mathematical expressions.

In [1]:
```
from sympy import *
x = Symbol('x')
```

In [5]:
```
f=eval(input("The first function is "))
g=eval(input("The second function is "))

a=lambda x: f
b=lambda x: g

print("The addition of the functions gives: ",a(x)+b(x))
print("The subtraction of the functions gives: ",a(x)-b(x))
print("The multiplication of the functions gives: ",a(x)*b(x))
print("The division of the functions gives: ",a(x)/b(x))
print("f(g(x)): ",a(b(x)))
print("g(f(x)): ",b(a(x)))
```

```
The first function is 2*x
The second function is x**2+3*x+4
The addition of the functions gives:  x**2 + 5*x + 4
The subtraction of the functions gives:  -x**2 - x - 4
The multiplication of the functions gives:  2*x*(x**2 + 3*x + 4)
The division of the functions gives:  2*x/(x**2 + 3*x + 4)
f(g(x)):  2*x
g(f(x)):  x**2 + 3*x + 4
```

In [7]:
```
def userinput():

    a = input("Enter the first function: ")
    b = input("Enter the second function: ")

    f = lambda x: eval(a)
    g = lambda x: eval(b)

    print("f(2):", f(2))
    print("g(2):", g(2))
    print("Addition:", f(2)+g(2))
    print("Subtraction:", f(2)-g(2))
    print("Multiplication:", f(2)*g(2))
    print("Division:", f(2)/g(2))
    print("fog:", f(g(2)))
    print("gof:", g(f(2)))

userinput()
```

```
Enter the first function: 2*x
Enter the second function: 3*x**2+6*x-4
f(2): 4
g(2): 20
Addition: 24
Subtraction: -16
Multiplication: 80
Division: 0.2
fog: 40
gof: 68
```

Limits

1. Limit class is a SymPy class used to calculate the limits of functions.
2. S is also a special SymPy that contains special characters like infinity.

Syntax to find limit: Limit(expression, variable, value)

In [11]:
```
from sympy import Limit, S, Symbol
```

In [3]:
```
x=Symbol('x')
m=Limit(1/x,x,S.Infinity)
m
```

Out[3]: $\lim_{x \to \infty} \frac{1}{x}$

To obtain value use doit() function, for Limit function (capital L)

In [6]:
```
m.doit()
```

Out[6]: 0

To directly obtain value use limit() function (small l)

As the value of n increases, the value of the term (1+1/n)^n tends to e. Verify.

In [15]:
```
n=Symbol('n')
m=Limit((1+1/n)**n,n,S.Infinity)
m.doit()
```

Out[15]: e

Find limit x^2+2 as x tends to 2

In [16]:
```
x=Symbol('x')
m=Limit(x**2+2,x,2)
m.doit()
```

Out[16]: 6

Find limit x+1 as x tends to 2¶

In [18]:
```
x=Symbol('x')
m=Limit(x+1,x,2)
m.doit()
```

Out[18]: 3

Write a program to get the table of values from left side and right side

1. f(x)=x^2+2x+1

In [6]:
```
import numpy as np
n=float(input("Enter the value of the limit point: "))
print("x tend to n from left hand side")
l=np.linspace(n-0.1,n,10)
f1=l**2+2*l+1
print(f1)
print("x tends to n from right hand side")
m=np.linspace(n,n+0.1,10)
f2=m**2+2*m+1
f3=f2[::-1]
print(f3)


if f1[9]==f3[9]:
    print ("Limit exists and equals ",f1[9])
else:
    print("Limit does not exist")
```

```
Enter the value of the limit point: 9
x tend to n from left hand side
[ 98.01        98.23012346  98.45049383  98.67111111  98.89197531
  99.11308642  99.33444444  99.55604938  99.77790123 100.        ]
x tends to n from right hand side
[102.01       101.78567901 101.56160494 101.33777778 101.11419753
 100.8908642  100.66777778 100.44493827 100.22234568 100.        ]
Limit exists and equals  100.0
```

Conclusion:

Functions and various kinds of operations were defined and performed. Limits of functions were found out.

Topic 4- Solving calculus problems: continuity, and derivatives using python

Date - 15/2/22

Aim:

To explore the concepts of calculus such as continuity, derivatives and maxima and minima and solve questions regarding the same using python.

Code:

Write codes to check about continuity of a function at a point

In [10]:
```
from sympy import Limit, S, Symbol
x=Symbol('x')
m=Limit(1/x,x,S.Infinity)
m.doit()
```

Out[10]: 0

In [18]:
```
## Finding right hand limit
from sympy import Limit, S, Symbol
x=Symbol('x')
rlim=Limit(1/x,x,0,dir='+').doit
print(rlim)
```

```
<bound method Limit.doit of Limit(1/x, x, 0)>
```

In [13]:
```
## Finding left hand limit
from sympy import Limit, S, Symbol
x=Symbol('x')
llim=Limit(1/x,x,0,dir='-').doit
print(llim)
```

```
<bound method Limit.doit of Limit(1/x, x, 0, dir='-')>
```

In [15]:
```
if rlim==llim==1/0:
    print ("Limit exists and equals ",rlim)
else:
    print("Limit does not exist")
```

```
Limit does not exist
```

In [16]:
```
## Alternative method

import sympy as sp
x=sp.Symbol('x')
f=1/x
value=0
def checkifcontinuous (func,x,symbol):
    return (sp.limit(func,x,symbol).is_real)
print(checkifcontinuous(f,x,value))
```

```
False
```

Check existence of limit and continuity for a given value

In [10]:
```
import numpy as np
import sympy as sp
n=float(input("Enter the value of the limit point: "))
print("x tend to n from left hand side")
x=sp.Symbol('x')
eq=x**2+2*x+1
l=np.linspace(n-0.1,n,10)
f1=l**2+2*l+1
print(f1)
print("x tends to n from right hand side")
m=np.linspace(n,n+0.1,10)
f2=m**2+2*m+1
f3=f2[::-1]
print(f3)


if round(f1[9])==round(f3[9]):
    print ("Limit exists and equals ",f1[9])
else:
    print("Limit does not exist")

print("The value of f(x) for x=n is ",eq.subs(x,n))

if round(f1[9])==eq.subs(x,n):
    print ("Continuity exists because LHL=RHL=f(a)")
else:
    print("Continuity does not exist")
```

```
Enter the value of the limit point: 9
x tend to n from left hand side
[ 98.01        98.23012346  98.45049383  98.67111111  98.89197531
  99.11308642  99.33444444  99.55604938  99.77790123 100.        ]
x tends to n from right hand side
[102.01       101.78567901 101.56160494 101.33777778 101.11419753
 100.8908642  100.66777778 100.44493827 100.22234568 100.        ]
Limit exists and equals  100.0
The value of f(x) for x=n is  100.000000000000
Continuity exists because LHL=RHL=f(a)
```

Derivatives

The derivative of a function y=f(x) expresses the rate of change in dependent variable y with respect to independent variable x. Denoted by f'(x). Use the concept; lim(h->0) (f(x+h)-f(x))/h=f'(x) [First principle definition]

Example 1: Find derivative for S(t)=5t^2+2t+8

In [5]:
```
import numpy as np
import sympy as sp
t=sp.Symbol('t')
st=5*t**2+2*t+8
from sympy import Limit, S, Symbol
lim=Limit(st,t,0)
lim.doit()
```

Out[5]: 8

In [8]:
```
from sympy import Limit, S, Symbol
h=Symbol('h')
t=Symbol('t')
p=Limit(((5*(t+h)**2+2*(t+h)+8)-(5*t**2+2*t+8))/h,h,0)
p.doit()
```

Out[8]: $10t + 2$

In [11]:
```
x=Symbol('x')
f=lambda t: 5*t**2+2*t+8
l=(f(x+h)-f(x))/h
lim=Limit(l,h,0)
lim.doit()
```

Out[11]: $10x + 2$

In [14]:
```
from sympy import Derivative
d=Derivative(st,t)
d ## Using Derivative function
```

Out[14]: $\frac{d}{dt}\left(5t^2 + 2t + 8\right)$

In [15]:
```
d.doit()
```

Out[15]: $10t + 2$

In [16]:
```
## To find at a particular point

d.doit().subs({t:1})
```

Out[16]: 12

Example 2: Find derivative for (x^3+x^2+x)(x^2+x)

In [21]:
```
eq=(x**3+x**2+x)*(x**2+x)
d1=Derivative(eq,x)
d1.doit()
```

Out[21]: $(2x + 1)\left(x^3 + x^2 + x\right) + \left(x^2 + x\right)\left(3x^2 + 2x + 1\right)$

Example 3: Find derivatives of the following functions

a) f(x)=2x-3/4 b) f(x)=x^5(3-6x^-9) c) sinxcosx d) 2tanx-7secx

In [22]:
```
## a) f(x)=2x-3/4
eq2=2*x-3/4
d2=Derivative(eq2,x)
d2.doit()
```

Out[22]: 2

In [23]:
```
## b) f(x)=x^5(3-6x^-9)
eq3=x**5*(3-6*x**-9)
d3=Derivative(eq3,x)
d3.doit()
```

Out[23]: $5x^4\left(3-\frac{6}{x^9}\right)+\frac{54}{x^5}$

In [29]:
```
## c) sinxcosx
import sympy
import numpy as np
eq4=sympy.sin(x)*sympy.cos(x)
d4=Derivative(eq4,x)
d4.doit()
```

Out[29]: $-\sin^2(x)+\cos^2(x)$

In [30]:
```
## d) 2tanx-7secx
eq4=2*sympy.tan(x)-7*sympy.sec(x)
d4=Derivative(eq4,x)
d4.doit()
```

Out[30]: $2\tan^2(x)-7\tan(x)\sec(x)+2$

Write a program to find derivative of any function from the user.

In [42]:
```
import sympy
a = input("Enter the function: ")
Der=Derivative(a,x)
Der.doit()
```

```
Enter the function: 2*x**3+7*x
```

Out[42]: $6x^2+7$

In [1]:
```
# Variable as anything
from sympy import *
import math
import sympy
var=Symbol(input("Enter a variable to differentiate with : "))
user=input("Enter a function with chosen variable and y : ")
Derivative(user,var).doit()
```

```
Enter a variable to differentiate with : t
Enter a function with chosen variable and y : t**2+3*t+9
```

Out[1]: $2t + 3$

In [3]:
```
from sympy import *
import math
import sympy
from sympy import sympify, pprint
from sympy.core.sympify import SympifyError

def derivative(f, var):

    var = Symbol(var)

    d = Derivative(f,var).doit()

    print(d)



if __name__=='__main__':

    f = input('Enter a function: ')

    var = input("Enter the Variable to differentiate wrt: ")

    try:

        f = sympify(f)

    except SympifyError:

        print("Invalid Input")

    else:

        Derivative(f, var)
```

```
Enter a function: sin(x)
Enter the Variable to differentiate wrt: x
```

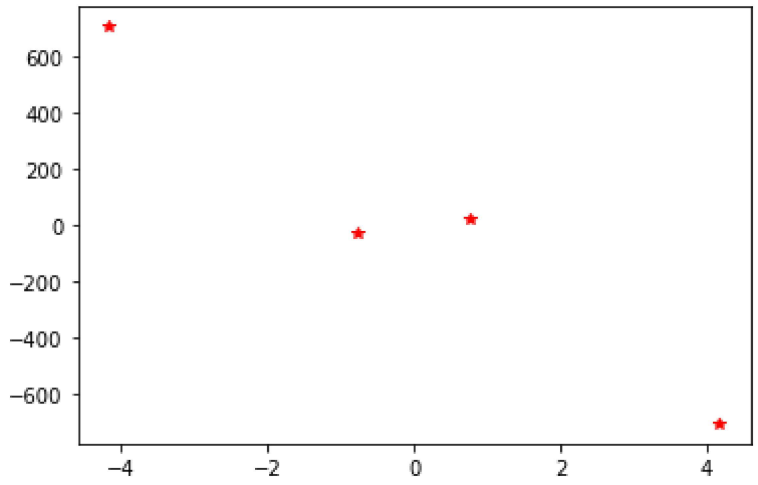
Maxima and minima

Find maxima and minima of the following function x^5-30x^3+50x

In [3]:
```
from sympy import Symbol,solve,Derivative

x=Symbol('x')
d=x**5-30*x**3+50*x
D1=Derivative(d,x).doit()
criticalpoints=solve(D1)
D2=Derivative(d,x,2).doit()

for i in range(len(criticalpoints)):
    print("critical points",i+1,"=",criticalpoints[i])
    print("Second Derivative at ",criticalpoints[i],"=",D2.subs({x:criticalpoints
```

```
critical points 1 = -sqrt(9 - sqrt(71))
Second Derivative at  -sqrt(9 - sqrt(71)) = 127.661060789073

critical points 2 = sqrt(9 - sqrt(71))
Second Derivative at  sqrt(9 - sqrt(71)) = -127.661060789073

critical points 3 = -sqrt(sqrt(71) + 9)
Second Derivative at  -sqrt(sqrt(71) + 9) = -703.493179468151

critical points 4 = sqrt(sqrt(71) + 9)
Second Derivative at  sqrt(sqrt(71) + 9) = 703.493179468151
```

In [4]:
```
## Plot the curve at critical points

from  matplotlib.pyplot import *
from numpy import *
a=linspace(-5,5)
b=a**5-30*a**3+50*a

for i in range(len(criticalpoints)):
    plot(criticalpoints[i],d.subs({x:criticalpoints[i]}),color="r",marker="*")
```



Conclusion:

The concepts of continuity of a function, existence of limit, derivatives of functions as well as maxima and minima of functions were explored in this section.

Topic 5:- Application of derivatives: cost function, revenue function, marginal cost, marginal revenue.

Date - 7/3/22

Aim:

To explore the concept of cost function, revenue function and the profit function. Further, to solve questions involving these concepts using maths and python.

APPLICATIONS OF DERIVATIVES

Management whether or not it knows calculus, utilises many functions of the sort we have been considering Examples of such functions are:

- Cost Function
- Revenue Function
- Profit Function

In [ ]:

COST FUNCTION

Suppose $C(x)$ is the total cost that a company incurs in producing x units of a commodity. The function C is called the cost function.

If the number of items produced is increased from x_1 to x_2, then the additional cost is $\Delta C = C(x_2) - C(x_1)$ and average rate of change of cost is $\Delta C/\Delta x$ (or) $\frac{\Delta C}{\Delta x}$.

The limit of this quantity as $\Delta x \to 0$, that is instantaneous rate of chane of cost w.r.t number of items produced is called **Marginal cost**.

REVENUE FUNCTION

In general, a business is concerned not only with its costs, but also with its revenues. Now let's consider marketing.

Let $p(x)$ be the price per unit that the company can cahnge if it sells x units. Then p is called **demand function** or **price function**.

If x units are sold and the price per unit is $p(x)$, then the total revenue is

$$R(x) = xp(x)$$

and R is called the **revenue function**.

The derivative of the revenue function is called the **marginal revenue function** and is the rate of change of revenue w.r.t number of units sold.

PROFIT FUNCTION

If x unitd are sold, the the total profit is

$$P(x) = R(x) - C(x)$$

and P is called the **profit function**.

The **marginal profit function** is P', the derivative of the profit function.

Code:

In [4]:
```
# Question for Cost Function
#SOURCE: https://study.com/academy/practice/quiz-worksheet-cost-function-in-calcu
```

The cost function for producing light bulbs is shown below.

$$C(x) = .001x + .0000000025x^2 + 100,000$$

a) Find the total cost of producing 2 million lightbulbs.

b) The firm wants to increase its production to 2.75 million lightbulbs. What is the additional cost and the average rate of change of cost?

In [6]:
```
c = lambda x: .001*x + .0000000025*x**2 + 100000
x1 = 2000000
c1 = c(x1)
print(f"The cost of producing 2 million lightbulbs is ${c1}.")
```

```
The cost of producing 2 million lightbulbs is $112000.0.
```

In [7]:
```
x2 = 2750000
c2 = c(x2)
change = c2 - c1
print(f"The increase in cost for the firm is ${change}.")
```

```
The increase in cost for the firm is $9656.25.
```

In [8]:
```
avg_cost = (c2 - c1) / (x2 - x1)
print(f"The average cost for the firm is ${avg_cost} per unit.")
```

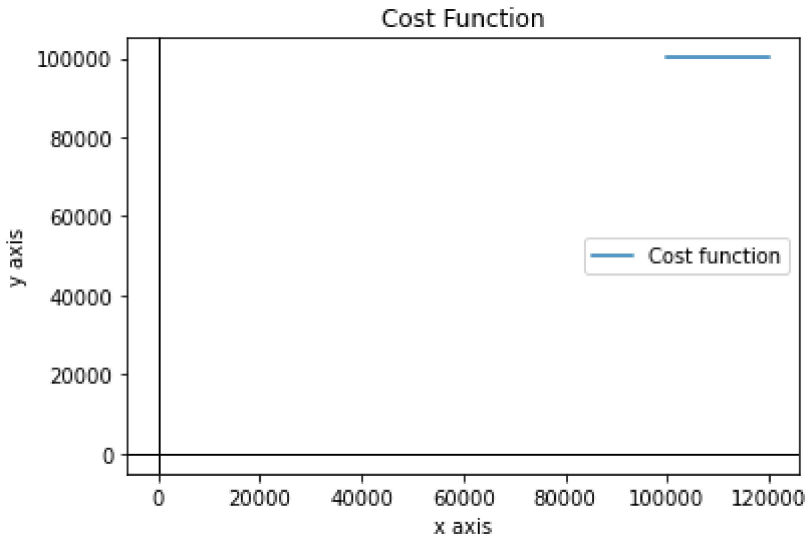
```
The average cost for the firm is $0.012875 per unit.
```

In [9]:
```
from sympy import Symbol, Derivative
x = Symbol('x')
marginal_cost = Derivative(c(x),x,1).doit()
print(f"The marginal cost is ({marginal_cost}) per additional unit.")
```

```
The marginal cost is (5.0e-9*x + 0.001) per additional unit.
```

In [10]:
```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(100000, 120000, 1000)
plt.plot(x, .001*x + .0000000025*x**2 + 100000, label='Cost function')
plt.xlabel('x axis')
plt.ylabel('y axis')
plt.title("Cost Function")
plt.axhline(0,lw=1,color='black')
plt.axvline(0,lw=1,color='black')
plt.legend()
```

Out[10]:
```
<matplotlib.legend.Legend at 0x245fca4d160>
```



In [11]:
```
# Question for Revenue Function
#SOURCE: https://www.calculushowto.com/cost-revenue-and-profit-function-examples/
```

A lemonade stand sold lemonades at a price of $0.50 each.

The Revenue function of the stand will be

$$R = 0.50 * x$$

- Find the revenue of selling 50 lemonades.
- Find the marginal revenue of selling an additional unit.

In [12]:
```
#The revenue function
r = lambda x: 0.50*x
x1= 50
r1= r(x1)
print(f"The revenue earned from selling 50 lemonades is ${r1}")
```

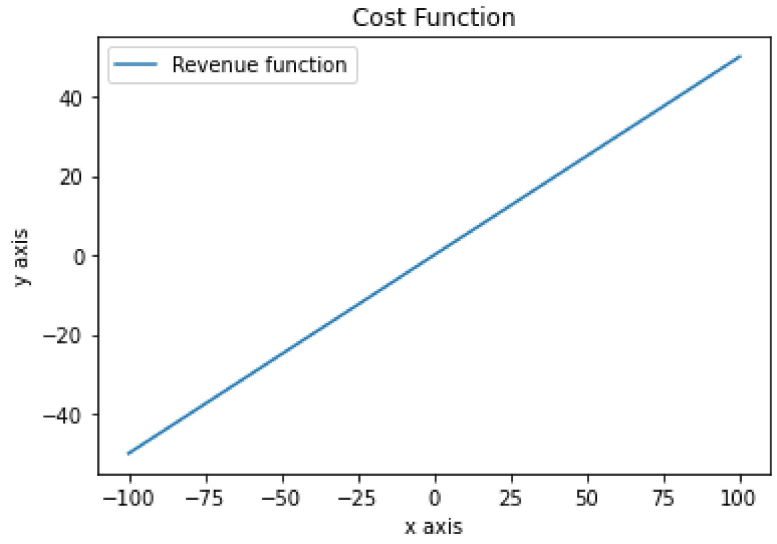
```
The revenue earned from selling 50 lemonades is $25.0
```

In [13]:
```
from sympy import *
x= Symbol('x')
marginal_revenue = Derivative(r(x),x,1).doit()
print(f"The marginal revenue is {marginal_revenue} for an additional unit")
```

```
The marginal revenue is 0.500000000000000 for an additional unit
```

In [14]:
```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(-100, 100, 1000)
plt.plot(x, 0.5*x , label='Revenue function')
plt.xlabel('x axis')
plt.ylabel('y axis')
plt.title("Cost Function")
plt.legend()
```

Out[14]: <matplotlib.legend.Legend at 0x245fd1e7d60>



In [ ]:
```
# Question for Profit Function
#SOURCE: http://www.math.utep.edu/faculty/cmmundy/Math%201320/Cost%20examples.pdf
```

Your college newspaper, The Collegiate Investigator, has fixed production costs of 70 per edition, and marginal printing and distribution costs of 40/copy. The Collegiate Investigator sells for 50/copy.

a) Write down the associated cost, revenue, and profit functions.

b) What profit (or loss) results from the sale of 500 copies of The Collegiate Investigator?

c) How many copies should be sold in order to break even?

In [15]:
```
import numpy
from numpy import *
import math
import sympy
from sympy import *

x= Symbol('x')
# a)
C= lambda x: 0.40*x+70
R= lambda x: 0.50*x

#Since profit= revenue -cost
P= R(x)-C(x)
P
```

Out[15]: $0.1x - 70$

In [16]:
```
# b) the profit/loss that results from selling 500 copies is P.
P.subs({x:500})
```

Out[16]: -20.0

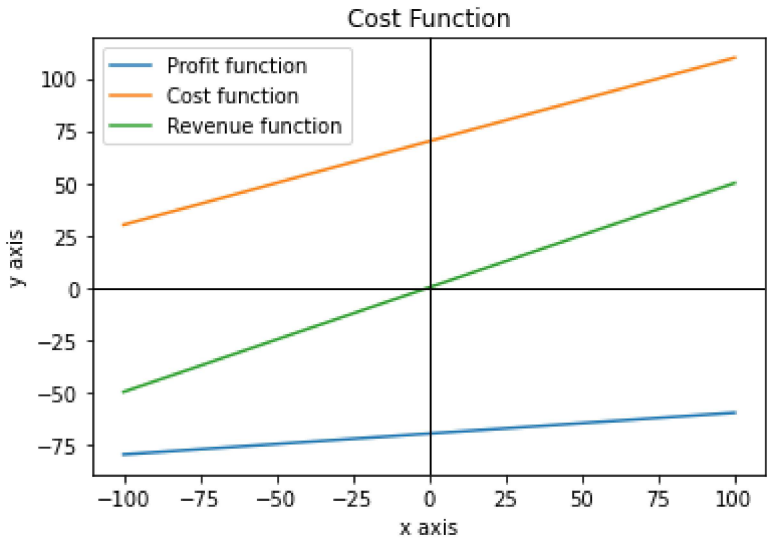
In [17]:
```
# This means that by selling 500 copies, they incur a loss of $20.
```

In [18]:
```
# c) For break even quantity, we set profit equal to zero. Now,
print("The break even quantity is:", solve(P,x=0))
```

```
The break even quantity is: [700.000000000000]
```

In [19]:
```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(-100, 100, 1000)
plt.plot(x, 0.1*x-70 , label='Profit function')
plt.plot(x, 0.4*x+70 , label='Cost function')
plt.plot(x, 0.5*x , label='Revenue function')
plt.xlabel('x axis')
plt.ylabel('y axis')
plt.title("Cost Function")
plt.axhline(0,lw=1,color='black')
plt.axvline(0,lw=1,color='black')
plt.legend()
```

Out[19]:
```
<matplotlib.legend.Legend at 0x245fd462610>
```



Conclusion:

The application of maths, specifically of calculus and derivatives was used in solving questions regarding cost, profit and revenue function.

Topic 6:- Solving differential equations using Sympy

Date - 22/3/22

Aim:

To solve differential equations using dsolve function. Further to find particular solutions using the given initial conditions.

Code:

Note: dsolve() solves any supported kind of ordinary differential equation.

The syntax is as follows: dsolve(eq,f(x))

Q1) Solve the differential equation dy/dx=1

In [1]:
```
from sympy import *
x=Symbol('x') ## defining independent variable
y=Function('y')(x) ## defining dependent variable
d=Eq(y.diff(x),1) ## assigning differential equation
print(d) ## presented in a straight line
display(d) ## presented in equation form
dsolve(d,y)
```

```
Eq(Derivative(y(x), x), 1)
```

$$\frac{d}{dx}y(x) = 1$$

Out[1]: $$y(x) = C_1 + x$$

In SymPy, an expression not in an Eq is automatically assumed to be equal to 0.

Q2) Solve the differential equation 6y+5y'+y''=0

In [2]:
```
from sympy import *
x=Symbol('x') ## defining independent variable
y=Function('y')(x) ## defining dependent variable
d1=Eq(6*y+5*y.diff(x)+y.diff(x,2),0) ## assigning differential equation
display(d1) ## presented in equation form
display("The solution of the given differential equation is ",dsolve(d1,y))
```

$$6y(x) + 5\frac{d}{dx}y(x) + \frac{d^2}{dx^2}y(x) = 0$$

```
'The solution of the given differential equation is '
```

$$y(x) = \left(C_1 + C_2 e^{-x}\right) e^{-2x}$$

Q3) Solve the differential equation y+y'''=0

In [3]:
```
from sympy import *
d2=y+y.diff(x,3) ## assigning differential equation without Eq (just use minus RH
display(d2)
display("The solution of the given differential equation is ",dsolve(d2,y))
```

$$y(x) + \frac{d^3}{dx^3}y(x)$$

'The solution of the given differential equation is '

$$y(x) = C_3 e^{-x} + \left(C_1 \sin\left(\frac{\sqrt{3}x}{2}\right) + C_2 \cos\left(\frac{\sqrt{3}x}{2}\right)\right) e^{\frac{x}{2}}$$

Solving using initial conditions and finding particular solutions

1) Solve the differential equation y'-1=0, given y(0)=1

In [4]:
```
from sympy import *
x=Symbol('x')
y=Function('y')(x)
f= diff(y,x,1)-diff(x)
print(f)
display(f)
sol=dsolve(f,y)
display(sol)
```

```
Derivative(y(x), x) - 1
```

$$\frac{d}{dx}y(x) - 1$$

$$y(x) = C_1 + x$$

In [5]:
```
C1=Symbol('C1')
constants=solve(sol.subs({x:0,y:1}))
constants
sol.subs(C1,constants[0])
```

Out[5]: $y(x) = x + 1$

2) Solve dP(t)/dt=kP(t) using initial conditions P(0)=18 and P (2)=39.

In [6]:
```
from sympy import *
k=Symbol('k')
t=Symbol('t')
p=Function('p')(t)
f= diff(p,t,1) - k*p
display(f)
sol=dsolve(f,p)
display(sol)
```

$$-kp(t) + \frac{d}{dt}p(t)$$

$$p(t) = C_1 e^{kt}$$

In [7]:
```
C1=Symbol('C1')
constants=solve(sol.subs({t:0,p:18}))
constants
man=sol.subs(C1,constants[0])
display(man)
```

$$p(t) = 18e^{kt}$$

In [8]:
```
k=Symbol('k')
constants=solve(man.subs({t:2,p:39}),k)
print(constants)
```

```
[-log(6) + log(78)/2, -log(6) + log(78)/2 + I*pi]
```

In [11]:
```
print ("The two Particular solutions are ")
display(sol.subs({k:constants[0],C1:18}))
```

```
The two Particular solutions are
```

$$p(t) = 18e^{t\left(-\log(6)+\frac{\log(78)}{2}\right)}$$

In [12]:
```
display(sol.subs({k:constants[1],C1:18}))
```

$$p(t) = 18e^{t\left(-\log(6)+\frac{\log(78)}{2}+i\pi\right)}$$

Conclusion:

Sympy package and dsolve function in python can be used to solve differential equations and further help to find general and particular solutions.

Topic 7:- Construction of slope fields of Ordinary differential equations of form fields

Date - 5/4/22

Aim:

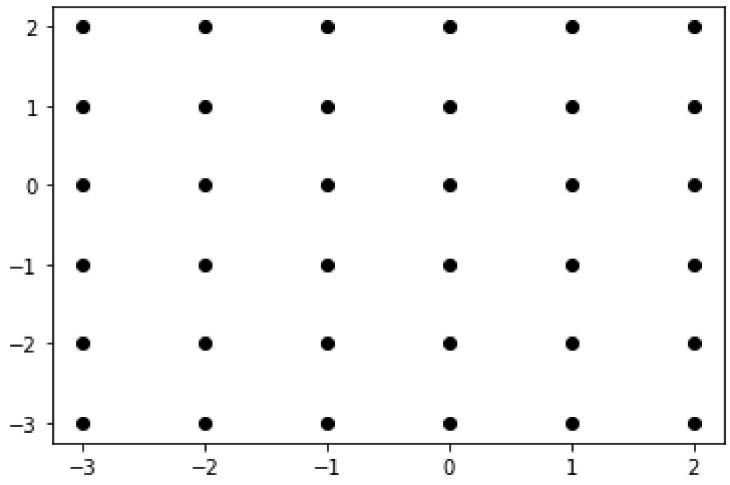
To explore the concepts of construction of slope fields for different differential equations using meshgrid. Further to also plot stream plots and scalar fields.

Code:

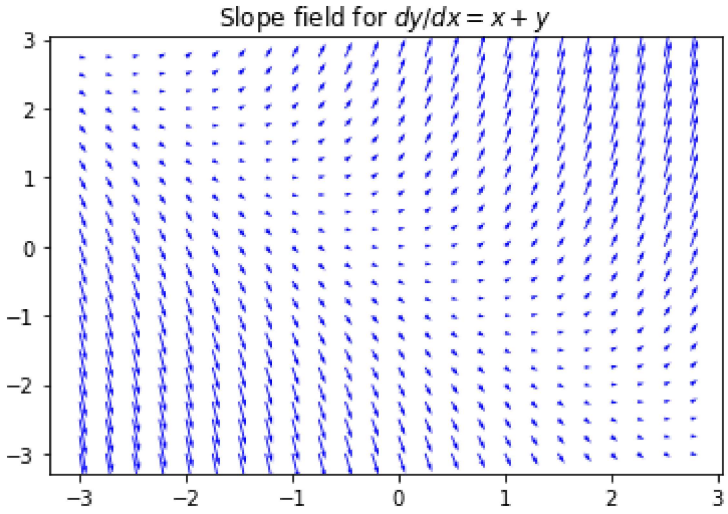
1. Plot the slope field for $dy/dx = x + y$

In [13]:
```
import numpy as np
import matplotlib.pyplot as plt
```

In [14]:
```
x=np.arange(-3,3)
y=np.arange(-3,3)
X,Y=np.meshgrid(x,y)
plt.plot(X,Y,'o',color='black')
None
```

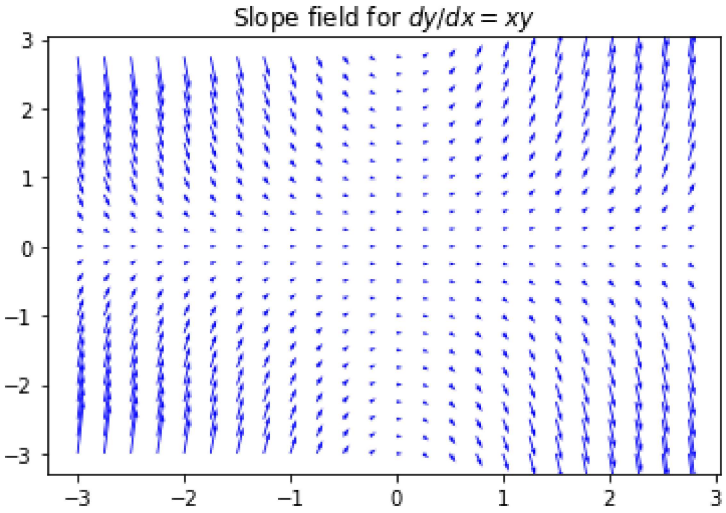


In [15]:
```
x=np.arange(-3,3,0.25)
y=np.arange(-3,3,0.25)
X,Y=np.meshgrid(x,y)

U=1
V=X+Y

plot=plt.figure()
plt.quiver(X,Y,U,V,color='Blue') ## shows vector lines as arrows
plt.title("Slope field for $dy/dx=x+y$")

plt.show(plot)
```



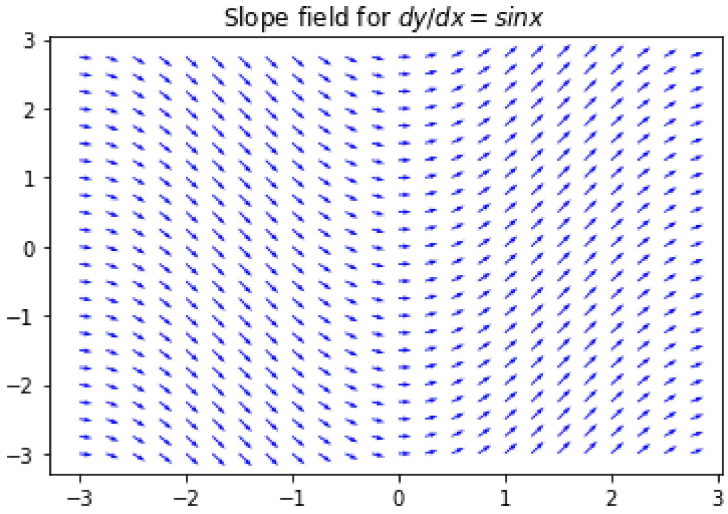
2. Plot the slope field for $dy/dx=xy$

In [16]:
```
x=np.arange(-3,3,0.25)
y=np.arange(-3,3,0.25)
X,Y=np.meshgrid(x,y)

U=np.ones(V.shape)
V=X*Y

plot=plt.figure()
plt.quiver(X,Y,U,V,color='Blue') ## shows vector lines as arrows
plt.title("Slope field for $dy/dx=xy$")

plt.show(plot)
```



3. Plot the slope field for dy/dx=sin(x)

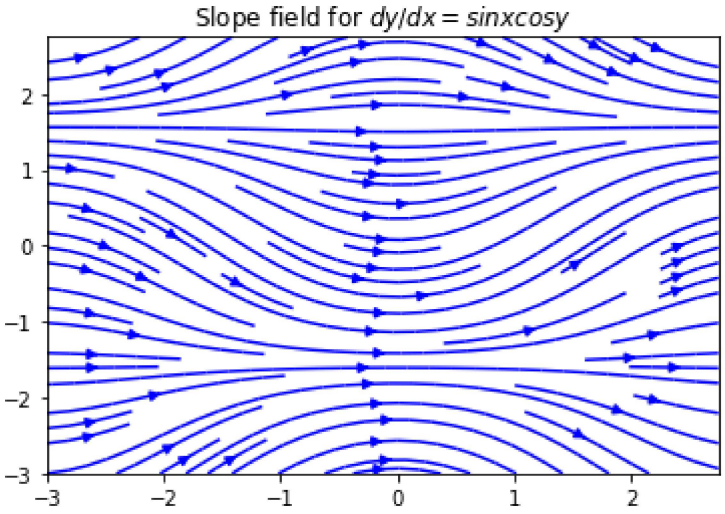
In [17]:
```
x=np.arange(-3,3,0.25)
y=np.arange(-3,3,0.25)
X,Y=np.meshgrid(x,y)

U=1
V=np.sin(X)

plot=plt.figure()
plt.quiver(X,Y,U,V,color='Blue') ## shows vector lines as arrows
plt.title("Slope field for $dy/dx=sinx$")

plt.show(plot)
```



Streamplots

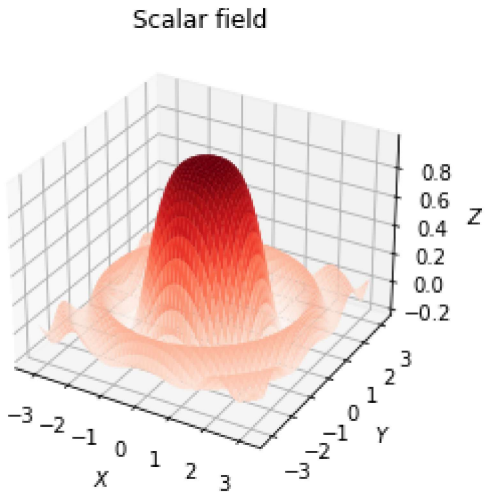
Plot the slope field for $dy/dx = sinxcosy$

In [18]:
```
x=np.arange(-3,3,0.25)
y=np.arange(-3,3,0.25)
X,Y=np.meshgrid(x,y)

U=np.ones(V.shape)
V=np.sin(X)*np.cos(Y)

plot=plt.figure()
plt.streamplot(X,Y,U,V,color='Blue', density=1,arrowsize=1,cmap="spring") ## show
plt.title("Slope field for $dy/dx=sinxcosy$")

plt.show(plot)
```



Plot the scalar field for $f(x, y) = (sin(x^2 + y^2))/(x^2 + y^2)$

In [19]:
```
from mpl_toolkits import mplot3d
import numpy as np
import matplotlib.pyplot as plt

## Constructing scalar field
x=np.linspace(-np.pi,np.pi,200)
y=np.linspace(-np.pi,np.pi,200)
X,Y=np.meshgrid(x,y)

Z=(np.sin(X**2+Y**2))/(X**2+Y**2)

fig=plt.figure()

ax=plt.axes(projection='3d')
abc=ax.plot_surface(X,Y,Z,cmap="Reds")

ax.set_xlabel('$X$')
ax.set_ylabel('$Y$')
ax.set_zlabel('$Z$')

ax.set_title('Scalar field')
plt.show()
```



Conclusion:

Slope fileds were plotted using meshgrids. Also, concept of streamplot and scalar field was explored.

Topic 8- Mathematical model- Interest rates

Date - 22/4/22

Aim:

To find prinicipal value, interest rate and time periods given for the given questions using python.

Code:

1. Find simple and compund interest, by taking inputs of principal amount, interest rate and time

In [21]:
```
p=int(input("Enter a principal amount (Rs.): "))
r=int(input("Enter rate of interest (%): "))
t=int(input("Enter a time interval (years): "))
si=(p*r*t)/100
print("The simple interest is Rs. ",si)
print("THe total amount is Rs.",si+p)
```

```
Enter a principal amount (Rs.): 1000
Enter rate of interest (%): 5
Enter a time interval (years): 25
The simple interest is Rs.  1250.0
THe total amount is Rs. 2250.0
```

In [22]:
```
p=int(input("Enter a principal amount (Rs.): "))
r=int(input("Enter rate of interest (%): "))
t=int(input("Enter a time interval (years): "))
ci=p*(1+r/100)**t-p
print("The compound interest is Rs.",ci) ## compounded annually
print("THe total amount is Rs.",ci+p)
```

```
Enter a principal amount (Rs.): 10000
Enter rate of interest (%): 20
Enter a time interval (years): 30
The compound interest is Rs. 2363763.1379976957
THe total amount is Rs. 2373763.1379976957
```

For compounding continuously, Pe^rt is the formula

2. How long does it take for a given amount of money to double at 10% per annum, compunded a) annually b) continuously

In [23]:
```
## annually
from sympy import *
P=Symbol('P')
t=Symbol('t')
x=Symbol('x')
R=0.1
x=2*P-P*(1+R)**t
solve(x,t)
```

Out[23]: [7.27254089734172]

In [24]:
```
## continuously
from sympy import *
P=Symbol('P')
t=Symbol('t')
x=Symbol('x')
R=0.1
x=2*P-P*exp(R*t)
solve(x,t)
```

Out[24]: [6.93147180559945]

3. In a bank, principle increases continuously at the rate of 5% per year. An amount of Rs.1000 is deposited with the bank, how much will it worth after 10 years.

In [25]:
```
from sympy import *
A=1000
t=10
r=0.05

Amount=A*exp(r*t)  ##Continuously Compounded
print('The Amount after 10 years will be equal to Rs.',Amount)
```

```
The Amount after 10 years will be equal to Rs. 1648.72127070013
```

4. In a bank, principle increases continuously at the rate of r% per year. Find the value of r if Rs. 100 double itself in 10 years.

In [26]:
```
from sympy import *
A=100
r=Symbol('r')
t=10
n=1

Rate=solve(2*A-A*(exp(r/100*t)),r) ##Continuously Compounded
print('The rate of interest equals',float(Rate[0]),'%')
```

```
The rate of interest equals 6.931471805599453 %
```

5. How long does it take for a given amount of money to double at 15% per annum compounded continuously.

In [27]:
```
from sympy import *
A=Symbol('A')
r=0.15
n=1
t=Symbol('t')

X=2*A-A*exp(r*t)
n=solve(X,t)
print("The time taken to double the principal(Compounded Continuously) is ",round
```

```
The time taken to double the principal(Compounded Continuously) is  4.6 years
```

6. Steve deposited Rs.9000 in a bank account and 10 years later he closed out the account,which is worth Rs.18000. What is the annual interest rate over the 10 years?

In [28]:
```
from sympy import *
A=9000
r=Symbol('r')
t=10
n=1

R=solve(9000*(1+r/100)**t-18000,r)
print('The rate of interest equals',round(float(R[1]),2),"%")
```

```
The rate of interest equals 7.18 %
```

Conclusion:

Basic interest calculations, such as simple and compund interest rates were performed using Python.

Topic 9:- Application of Derivatives in Exponential and Logistic

Date - 26/4/22

Aim:

To explore the concept of exponential and logistics model and solve questions relating to the same.

Code:

Exponential growth models

A growth model of a population represents the dynamics where the population's growth rate increases over time, in proportion to the size of the record

dp/dt=kp, where dp/dt denotes the population growth and k is constant pf proportionality.

1. If a population of a town icreased from 50000 to 100000 in five years, then find the population of the town after ten years.

In [29]:
```
from sympy import *
from numpy import *
from math import *
t=Symbol('t') ## defining independent variable
p=Function('p')(t) ## defining dependent variable
k=Symbol('k',real=True)
d=Eq(p.diff(t),k*p) ## assigning differential equation
print(d) ## presented in a straight line
display(d) ## presented in equation form
dsolve(d,p)
C1=Symbol('C1')
```

```
Eq(Derivative(p(t), t), k*p(t))
```

$$\frac{d}{dt}p(t) = kp(t)$$

In [30]:
```
b=dsolve(d,p)
b
```

Out[30]: $p(t) = C_1 e^{kt}$

In [31]:
```
b.subs({p:50000,t:0})
```

Out[31]: $50000 = C_1$

In [32]:
```
c=b.subs(C1,50000)
c
```

Out[32]: $p(t) = 50000e^{kt}$

In [33]:
```
d=c.subs({p:100000,t:5})
d
```

Out[33]: $100000 = 50000e^{5k}$

In [34]:
```
solve(d,k)
```

Out[34]:
```
[log(2**(1/5))]
```

In [35]:
```
e=b.subs({C1:50000,k:log(2**(1/5))})
e
```

Out[35]: $p(t) = 50000e^{0.138629436111989t}$

In [36]:
```
f=e.subs({t:10})
f
```

Out[36]: $p(10) = 200000.0$

2. If a population of a city has doubled in the past 25 years from 1 lakh, when will the city's population reach 5 lakhs?

In [37]:
```
from sympy import *
from numpy import *
from math import *
t=Symbol('t',real=True) ## defining independent variable
p=Function('p')(t) ## defining dependent variable
k=Symbol('k',real=True)
d=Eq(p.diff(t),k*p) ## assigning differential equation
print(d) ## presented in a straight line
display(d) ## presented in equation form
dsolve(d,p)
C1=Symbol('C1')
```

```
Eq(Derivative(p(t), t), k*p(t))
```

$$\frac{d}{dt}p(t) = kp(t)$$

In [38]:
```
b=dsolve(d,p)
b
```

Out[38]: $p(t) = C_1 e^{kt}$

In [39]:
```
b.subs({p:100000,t:0})
```

Out[39]: $100000 = C_1$

In [40]:
```
c=b.subs(C1,100000)
c
```

Out[40]: $p(t) = 100000e^{kt}$

In [41]:
```
d=c.subs({p:200000,t:25})
d
```

Out[41]: $200000 = 100000e^{25k}$

In [42]:
```
solve(d,k)
```

Out[42]:
```
[log(2**(1/25))]
```

In [43]:
```
e=b.subs({C1:100000,k:log(2**(1/25))})
e
```

Out[43]: $p(t) = 100000e^{0.0277258872223978t}$

In [44]:
```
f=e.subs({p:500000})
f
```

Out[44]: $500000 = 100000e^{0.0277258872223978t}$

Logistics Growth Model

Exponential growth is not very sustainable as it depends on infinite amounts of resources.

Exponential growth may happen if there are few individuals and many resources, but as the resources get used up or population becomes large, the growth rate slows and an S shaped curve is formed. (The growth rate will plateau or level off)

The population size at which it levels off, which represents the maximum population size a particular environment can support, is called the carrying capacity.

Mathematical model for Logistic growth

$$dP/dt = kP(N - P)/N$$

where N=carrying capacity, population size=P and k is per capita growth rate

(N-P) tells us how many more individuals can be added to the population before the carrying capacity

(N-P)/N is the fraction of the carrying capacity that has not yet been used up.

The more carrying capacity used up, the more (N-P)/N will reduce the growth rate.

When the population is tiny P is very small compared to N. The (N-P/N) term becomes approximately (N/N) or 1 giving us back the exponential equation. Thus the population grows near-exponentially at first but levels off more and more as it approaches N.

Q1. A bacteria culture starts with 1,000 bacteria and doubles in size every 3 hours. Find an exponential model for the size of the culture as a function of time t in hours.

In [20]:
```
from sympy import *
from numpy import *
from math import *
t=Symbol('t')
p=Function('p')(t)
k=Symbol('k',real=True)
d=Eq(p.diff(t),k*p)
print(d)
display(d)
dsolve(d,p)
C1=Symbol('C1')
```

```
Eq(Derivative(p(t), t), k*p(t))
```

$$\frac{d}{dt}p(t) = kp(t)$$

In [21]:
```
b=dsolve(d,p)
b
```

Out[21]: $p(t) = C_1 e^{kt}$

In [22]:
```
b.subs({p:1000,t:0})
```

Out[22]: $1000 = C_1$

In [23]:
```
c=b.subs(C1,1000)
c
```

Out[23]: $p(t) = 1000e^{kt}$

In [24]:
```
d=c.subs({p:2000,t:3})
d
```

Out[24]: $2000 = 1000e^{3k}$

In [25]:
```
solve(d,k)
```

Out[25]:
```
[log(2**(1/3))]
```

In [26]:
```
e=b.subs({C1:1000,k:log(2**(1/3))})
print("The exponential model for the size of the culture as a function of time t
display(e)
```

```
The exponential model for the size of the culture as a function of time t in ho
urs is given by:
```

$$p(t) = 1000e^{0.231049060186648t}$$

Q2. Frog population projections. The frog population in a small pond grows exponentially. The current population is 85 frogs, and the relative growth rate is 18% per year. (a) Find a function that models the number of frogs after t years. (b) Find the projected population after 3 years. (c) Find the number of years required for the frog population to reach 600.

In [27]:
```
from sympy import *
from numpy import *
from math import *
t=Symbol('t')
p=Function('p')(t)
k=Symbol('k',real=True)
d=Eq(p.diff(t),k*p)
print(d)
display(d)
dsolve(d,p)
C1=Symbol('C1')
```

```
Eq(Derivative(p(t), t), k*p(t))
```

$$\frac{d}{dt}p(t) = kp(t)$$

In [28]:
```
b=dsolve(d,p)
b
```

Out[28]: $p(t) = C_1 e^{kt}$

a)

In [29]:
```
l=b.subs({C1:85,k:0.18})
l
```

Out[29]: $p(t) = 85e^{0.18t}$

b)

In [30]:
```
e=l.subs({t:3})
print(e)

print("THe projected frog population after 3 years is approximately ", 146)
```

```
Eq(p(3), 145.860583285713)
THe projected frog population after 3 years is approximately  146
```

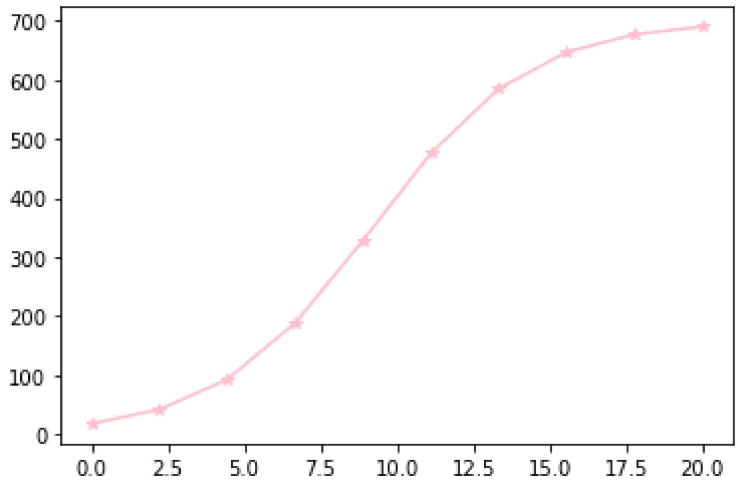
c)

In [31]:
```
import sympy
l=85*sympy.exp(0.18*t)-600
l
n=solve(l,t)
print("The number of years required for frog population to reach 600 is ",n[0],"
```

```
The number of years required for frog population to reach 600 is  10.8571022151
435  years
```

Q3. Researcher stock a tank with 18 and estimate the carrying capacity to be 700. The number of yeast increased by 0.395 times during the first year. Assuming that the yeast population growth satisfies the logistic equation, find population after t years.

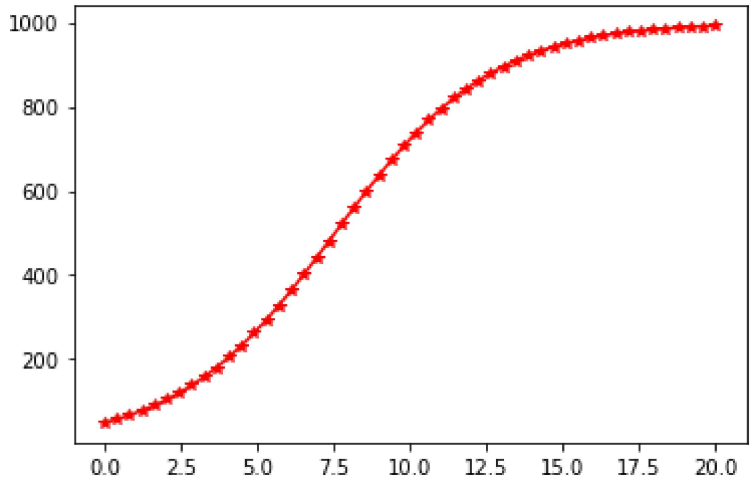
In [32]:
```
from scipy.integrate import odeint
import numpy as np
import matplotlib.pyplot as plt

def h(P,t):
    eq=k*P*(1-P/700)
    return eq
k=0.395
P0=18
t=np.linspace(0,20,10)
P2=odeint(h,P0,t)
plt.plot(t,P2,marker='*',color='pink')
None
```



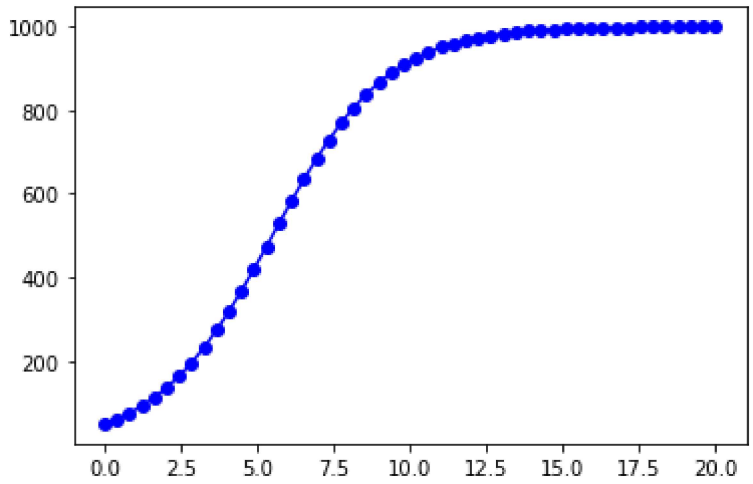
Q4. Biologists stock 50 bacterias and estimate the carrying capacity to be 1000. Assuming that the growth of bacteria satisfies the logistic equation, find population

after t years. For the following growth rates: a) 0.39 b) 0.535 c) 0.409.

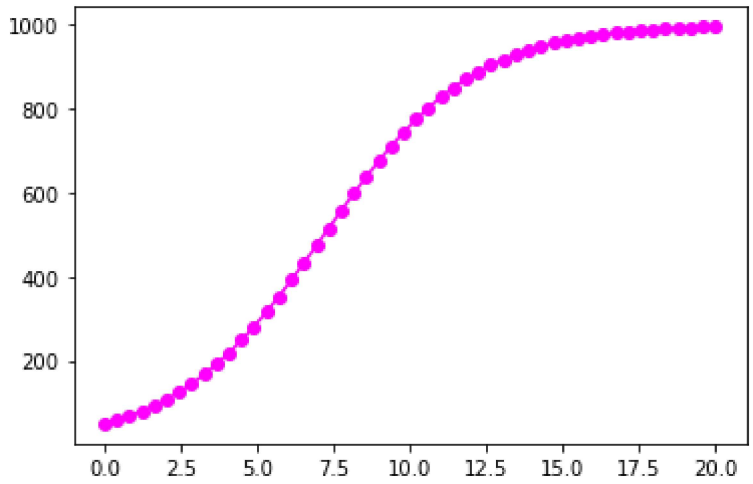
In [33]:
```
# a) 0.39

def h(P,t):
    eq=k*P*(1-P/1000)
    return eq
k=0.39
P0=50
t=np.linspace(0,20,50)
P2=odeint(h,P0,t)
plt.plot(t,P2,marker='*',color='red')
None
```



In [34]:
```
# b) 0.535

def h(P,t):
    eq=k*P*(1-P/1000)
    return eq
k=0.535
P0=50
t=np.linspace(0,20,50)
P2=odeint(h,P0,t)
plt.plot(t,P2,marker='o',color='blue')
None
```



In [35]:
```
# c) 0.409

def h(P,t):
    eq=k*P*(1-P/1000)
    return eq
k=0.409
P0=50
t=np.linspace(0,20,50)
P2=odeint(h,P0,t)
plt.plot(t,P2,marker='o',color='magenta')
None
```



Q5. The following table shows the number of yeast in laboratory

Time(hrs): 0,2,4,6,8,10,12,14,16,18

Yeast cells: 18,39,80,171,336,509,597,640,664,672

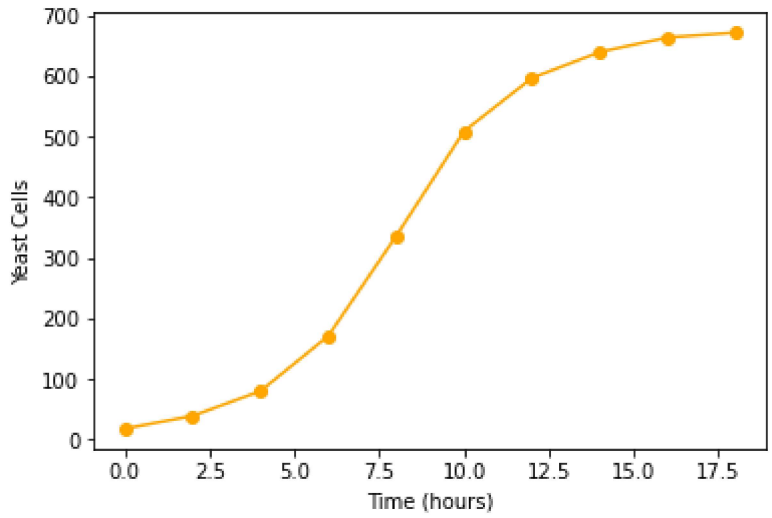
a. Plot the data and use the plot to estimate the carrying capacity for yeast population

b. Use data to estimate initial population growth.

c. Find both exponential and logistic model.

d. Compare predicted and observed values. Comment on how your model will fit the data.

In [36]:
```
time=[0,2,4,6,8,10,12,14,16,18]
yeast=[18,39,80,171,336,509,597,640,664,672]
plt.plot(time,yeast,marker='o',color='orange')
plt.xlabel('Time (hours)')
plt.ylabel('Yeast Cells')
None
```



Observing the graph, we can conclude that the carrying capacity is around 680.

In [37]:
```
from numpy import *
from sympy import *
from math import *
init_printing(use_unicode=True)
t=Symbol('t', real=True)
k=Symbol('k', real=True)
p=Function('p')(t)
d=Eq(p.diff(t),k*p)
display(d)
sol1=dsolve(d,p)
display(sol1)
C1=Symbol('C1')
sol1.subs({t:0, p:18})
```

$$\frac{d}{dt}p(t) = kp(t)$$

$$p(t) = C_1 e^{kt}$$

Out[37]: $18 = C_1$

In [38]:
```
c=sol1.subs({C1:18})
print('the particular solution is: ')
display(c)
```

the particular solution is:

$$p(t) = 18e^{kt}$$

In [39]:
```
print('the particular solution after substituting for p and t is: ')
d=c.subs({p:39, t:2})
display(d)
```

the particular solution after substituting for p and t is:

$$39 = 18e^{2k}$$

In [40]:
```
print('solving the particular solution for k we get: ')
s=solve(d,k)
display(s)
```

solving the particular solution for k we get:

$$\left[-\log(6) + \frac{\log(78)}{2}\right]$$

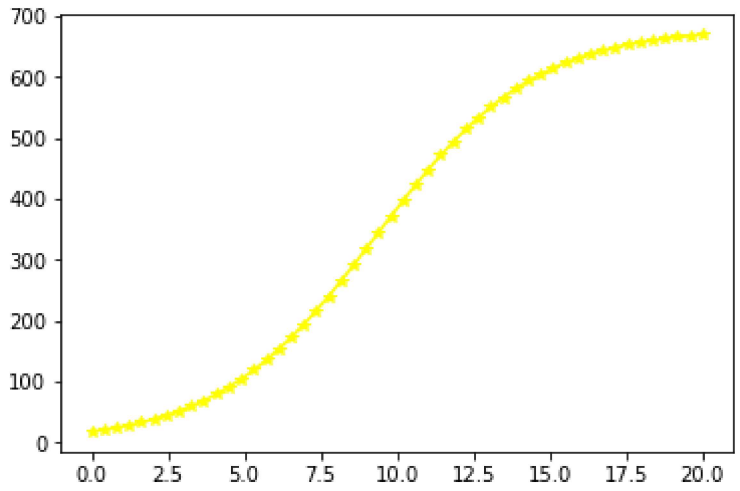
In [41]:
```
# c)
print('substituting the value of k in the particular solution we get: ')
e=sol1.subs({C1:18,k:-log(6)+log(78)/2})
display(e)
```

substituting the value of k in the particular solution we get:

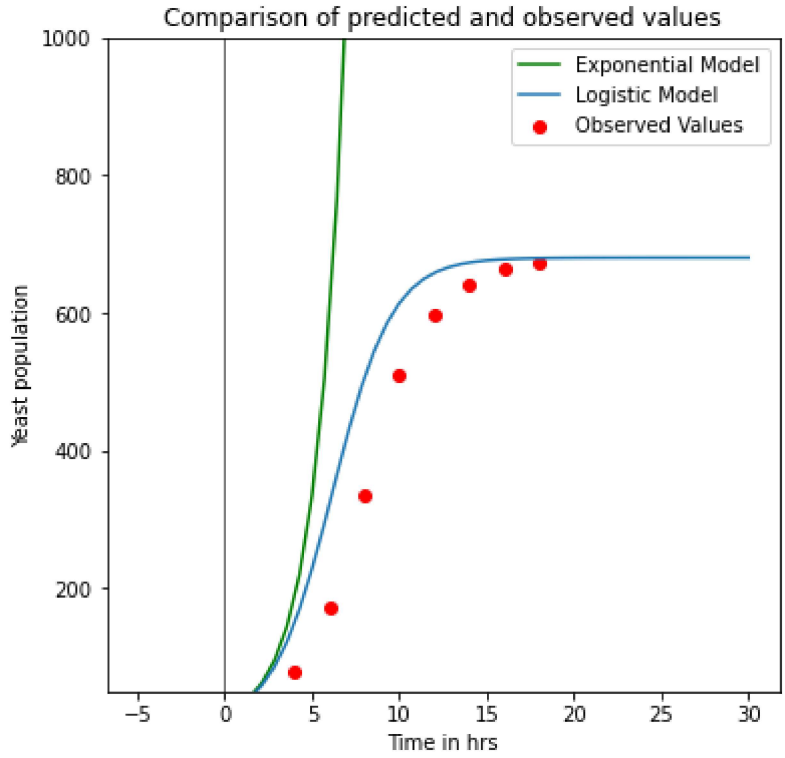
$$p(t) = 18e^{0.386594944116741t}$$

In [42]:
```
def h(P,t):
    eq=k*P*(1-P/680)
    return eq
k=-log(6)+log(78)/2
P0=18
t=np.linspace(0,20,50)
P2=odeint(h,P0,t)
plt.plot(t,P2,marker="*", color='yellow')
```

Out[42]: [<matplotlib.lines.Line2D at 0x245ff35e520>]



In [43]:
```
## d
X=np.linspace(-5,30)
Y=18*np.exp(7*X/12)
Z=680/((331/9)*np.exp(-7*X/12)+1)

plt.figure(figsize=(6,6))
plt.plot(X,Y,color='green',label='Exponential Model')
plt.plot(X,Z,label='Logistic Model')
plt.scatter(time,yeast,marker='o',label='Observed Values',color='red')

plt.ylim(ymin=50,ymax=1000)
plt.title('Comparison of predicted and observed values')
plt.xlabel('Time in hrs')
plt.ylabel('Yeast population')
plt.axhline(0,lw=0.5,color='black')
plt.axvline(0,lw=0.5,color='black')

plt.legend()
None
```



It can be observed that the exponential model is poor fit and that the logistics model varies more for the middle values however, it provides a good general fit.

Conclusion:

Logistics and exponential model was explored, and plotted for the given questions.

Topic 10:- Mathematical model: Simple pendulum and SIR model

Date - 30/4/22

Aim:

To explore the concept of the SIR model and Simple pendulum mathematical model. Simple pendulum has an integral usage in physics and the SIR model helps to assess the spread of diseases, for example the COVID-19 pandemic.

Code:

Mathematical function

$\mathbf{S(t)}$ = how many individuals, at time t, that have the possibility to get infected where t may count in hours/days.

These individuals make category called **susceptibles**, labelled as S.

$\mathbf{I(t)}$ = count how many there are that are infected.

These individuals make category called **infected**, labelled as I at time t.

An individual having recovered from the disease is assumed to gain immunity. There is also a small possibility that an infected will die. In either case, the individual is moved from the I category to a category we call the **removed category**, labeled with R.

$\mathbf{R(t)}$ = count the number of individuals in the R category at time t. Those who enter the R category, cannot leave this category.

Δt = changes that take place during a small time interval

$\mathbf{S \rightarrow I \rightarrow R}$

The SIR model describes the change in the population of each of these compartments in terms of two parameters, β and γ.

β describes the effective contact rate of the disease: an infected individual comes into contact with βN other individuals per unit time (of which the fraction that are susceptible to contracting the disease is $\frac{S}{N}$.

γ is the mean recovery rate: that is, $\frac{1}{\gamma}$ is the mean period of time during which an infected individual can pass it on.

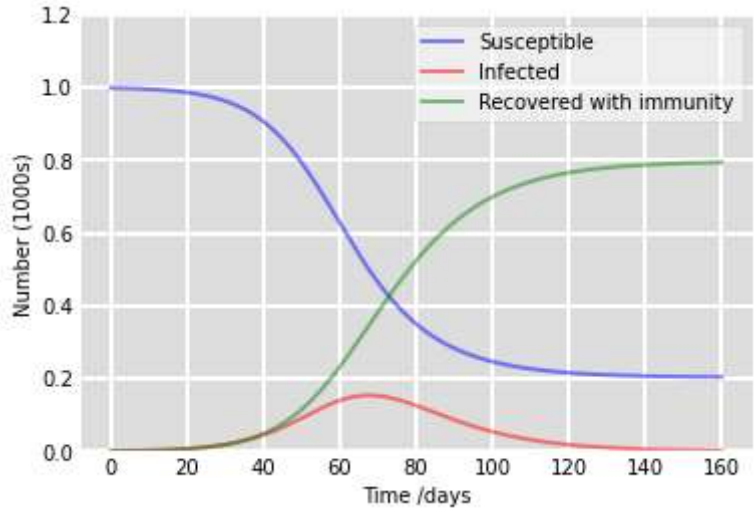
$\frac{dS}{dt} = -\frac{\beta SI}{N}$,

$\frac{dI}{dt} = \frac{\beta SI}{N} - \gamma I$,

$\frac{dR}{dt} = \gamma I$.

The following Python code integrates these equations for a disease characterised by parameters $\beta = 0.2$, $\frac{1}{\gamma}$ = 10 days in a population of N = 1000 (perhaps 'flu in a school). The model is started with a single infected individual on day 0: $I(0)$ = 1. The plotted curves of $S(t)$, $I(t)$ and $R(t)$ are styled to look a bit nicer than Matplotlib's defaults.

In [30]:
```
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt

# Total population, N.
N = 1000
# Initial number of infected and recovered individuals, I0 and R0.
I0, R0 = 1, 0
# Everyone else, S0, is susceptible to infection initially.
S0 = N - I0 - R0
# Contact rate, beta, and mean recovery rate, gamma, (in 1/days).
beta, gamma = 0.2, 1./10
# A grid of time points (in days)
t = np.linspace(0, 160, 160)

# The SIR model differential equations.
def deriv(y, t, N, beta, gamma):
    S, I, R = y
    dSdt = -beta * S * I / N
    dIdt = beta * S * I / N - gamma * I
    dRdt = gamma * I
    return dSdt, dIdt, dRdt

# Initial conditions vector
y0 = S0, I0, R0
# Integrate the SIR equations over the time grid, t.
ret = odeint(deriv, y0, t, args=(N, beta, gamma))
S, I, R = ret.T

# Plot the data on three separate curves for S(t), I(t) and R(t)
fig = plt.figure(facecolor='w')
ax = fig.add_subplot(111, facecolor='#dddddd', axisbelow=True)
ax.plot(t, S/1000, 'b', alpha=0.5, lw=2, label='Susceptible')
ax.plot(t, I/1000, 'r', alpha=0.5, lw=2, label='Infected')
ax.plot(t, R/1000, 'g', alpha=0.5, lw=2, label='Recovered with immunity')
ax.set_xlabel('Time /days')
ax.set_ylabel('Number (1000s)')
ax.set_ylim(0,1.2)
ax.yaxis.set_tick_params(length=0)
ax.xaxis.set_tick_params(length=0)
ax.grid(b=True, which='major', c='w', lw=2, ls='-')
legend = ax.legend()
legend.get_frame().set_alpha(0.5)
for spine in ('top', 'right', 'bottom', 'left'):
    ax.spines[spine].set_visible(False)
plt.show()
```

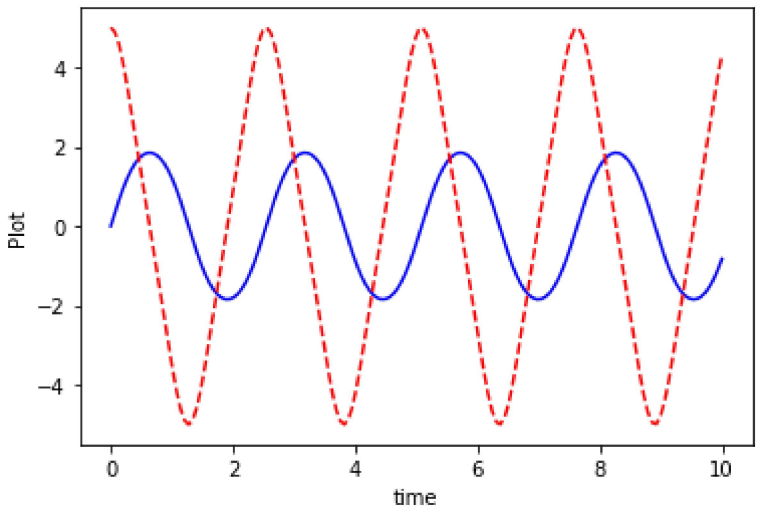


In [31]:
```
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt

# Total population, N.
N = 10000
# Initial number of infected and recovered individuals, I0 and R0.
I0, R0 = 1, 0
# Everyone else, S0, is susceptible to infection initially.
S0 = N - I0 - R0
# Contact rate, beta, and mean recovery rate, gamma, (in 1/days).
beta, gamma = 0.5, 1./20
# A grid of time points (in days)
t = np.linspace(0, 1600, 1600)

# The SIR model differential equations.
def deriv(y, t, N, beta, gamma):
    S, I, R = y
    dSdt = -beta * S * I / N
    dIdt = beta * S * I / N - gamma * I
    dRdt = gamma * I
    return dSdt, dIdt, dRdt

# Initial conditions vector
y0 = S0, I0, R0
# Integrate the SIR equations over the time grid, t.
ret = odeint(deriv, y0, t, args=(N, beta, gamma))
S, I, R = ret.T
print(ret.T)
```

```
[[ 9.99900000e+03  9.99836816e+03  9.99737711e+03 ...  4.54160054e-01
   4.54160054e-01  4.54160054e-01]
 [ 1.00000000e+00  1.56865218e+00  2.46057149e+00 ... -1.08969830e-12
  -1.08768396e-12 -1.08566962e-12]
 [ 0.00000000e+00  6.31928156e-02  1.62318392e-01 ...  9.99954584e+03
   9.99954584e+03  9.99954584e+03]]
```

Motion of a simple pendulum

In [32]:
```
import numpy as np
from scipy.integrate import odeint
import math
import matplotlib.pyplot as plt

## Defining function for ODE
def model(theta,t,b,g,l,m):
    theta1=theta[0]
    theta2=theta[1]
    dtheta_1=theta2
    dtheta_2=-(b/m)*theta2-(g/l)*math.sin(theta1)
    dtheta_dt=[dtheta_1, dtheta_2]
    return dtheta_dt

## Initializaing important parameters

b=0 # damping coefficient
l=1 # length of string
g=9.81 # acceleration due to gravity
m=0.1 # mass of bob
theta_0=[0,5] # intital condition
t=np.linspace(0,10,150) # time points

theta=odeint(model,theta_0,t,args=(b,g,l,m))

## Plotting results
plt.figure
plt.plot(t,theta[:,0],'b-')
plt.plot(t,theta[:,1],'r--')
plt.ylabel("Plot")
plt.xlabel("time")
plt.show()
None
```



Conclusion:

The SIR model and the Simple Pendulum Model was explored and plotted using python.

Topic 11:- Python Programming for Data Management

Date - 5/5/22

INTRODUCTION

Python has wide-ranging applications in all the fields especially in Finance and Banking. There is a need that systems at institutions like Banks, Libraries etc. get automated as manual work is impossible to conduct at large scale. Further to make registration, opening up of the account, checking balances and many other functions relating to bank, automation is required. This is where the Bank Management System is created using software like python.

Given below is an example of such a system. The code explains the following-

1. First, it takes as input customer name, customer ID, and their balance. Then it initialises deposit, withdrawl and balance variables as 0.

2. Next, the main menu or the many options avaible are presented to the user. These include the option of New Account, Withdraw, Deposit, Balance and Exit which as the name suggests allows the user to add a new account, Withdraw amount from the balance, Deposit amount in one's account, Check balance in one's account and finally to exit the system.

3. Different choices as mentioned above are coded as 1,2,3,4 and 5. Using these values different choices can be accessed by the user by inputing the same.

CODE

The code for executing the Bank Managment System as exaplined above is given below-

In [45]:
```
custName = ['Asheesh', 'Sidharth Manakil', 'Milan', 'Akshath']#this is the existi
custID = ['482', '943', '851', '497']#customer IDs are random
custBal = [98520, 85310, 74698, 2342]
deposit = 0
withdrawal = 0
balance = 0
start = 1 #this is the starting index for iterating through the list of names, ID
fin = 4 #this is the ending index for iterating through the list of names, IDs et
i = 0 #this value is used for iterating through the total number of customers tha

while True:

    print("          ----AAA Bank----       ")
    print("*************************************")
    print("***             MENU              ***")
    print("*** 1. New Account                 ***")
    print("*** 2. Withdraw                    ***")
    print("*** 3. Deposit                     ***")
    print("*** 4. Balance                     ***")
    print("*** 5. Exit                        ***")
    print("*************************************")

    choice = input("Please enter your choice : ")
    if choice == "1":
        print("You have selected 1")
        tot = eval(input("Please enter the number of customers you want to enter
        i = i + tot
        if i > 5:
            print("\n")
            print("Sorry. You cannot enter more than 5 customer details at a time
            i = i - tot
        else:
            while start <= i:
                name = input("Please enter the customer's full name : ")
                custName.append(name)
                pin = str(input("Please enter a secret number of your choice as P
                custID.append(pin)
                balance = 0
                deposit = eval(input("Please enter the amount you want to deposit
                balance = balance + deposit
                custBal.append(balance)
                print("\nName=", end=" ")
                print(custName[fin])
                print("Pin=", end=" ")
                print(custID[fin])
                print("Balance=", end=" ")
                print(custBal[fin], end=" ")
                print("-/Rs")
                start = start + 1
                fin = fin + 1
                print("\n Your account has been created successfully!-----")
                print("\n")
                print("Your name is avalilable on the customers list now : ")
                print(custName)
                print("\n")
                print("Note! Please do not lose your secret number")
```

```
                print("========================================")
        mainMenu = input("Press any key to return to the main menu")
    elif choice == "2":
        j = 0
        while j < 1:
            k = -1
            print("*** Withdrawal ***")
            name = input("Please enter your full name as mentioned while creating
            pin = input("Please enter your secret number : ")
            while k < len(custName) - 1: #this is -1 because the list starts from
                k = k + 1
                if name == custName[k]:
                    if pin == custID[k]:
                        j = j + 1
                        #print("Available Balance:", end=" ")
                        #print(custBal[k], end=" ")
                        #print("-/Rs\n")
                        balance = (custBal[k])
                        withdrawal = eval(input("Enter the amount you want to Wit
                        if withdrawal > balance:
                            print("You do not have sufficient balance. ")
                            print("Your Balance: ", balance, end=" ")
                        else:
                            balance = balance - withdrawal
                            print("\n")
                            print("----Withdraw Successfull!----")
                            custBal[k] = balance
                            print("Your New Balance: ", balance, end=" ")
            if j < 1:
                print("Your name and pin does not match!\n")
                break
        mainMenu = input("Press any key to return to the main menu")
    elif choice == "3":
        print("Deposit to Account")
        n = 0
        while n < 1:
            k = -1
            name = input("Please input name : ")
            pin = input("Please input pin : ")
            while k < len(custName) - 1:
                k = k + 1
                if name == custName[k]:
                    if pin == custID[k]:
                        n = n + 1
                        print("Your Current Balance: ", end=" ")
                        print(custBal[k], end=" ")
                        print("-/Rs")
                        balance = (custBal[k])
                        deposit = eval(input("Enter the value you want to deposit
                        balance = balance + deposit
                        custBal[k] = balance
                        print("\n")
                        print("----Deposit Complete----")
                        print("Your New Balance: ", balance, end=" ")
                        print("-/Rs\n\n")
            if n < 1:
                print("Your name and pin does not match!\n")
```

```
                break
        mainMenu = input("Press any key to return to the main menu.")
    elif choice == "4":
        print("Account Balance. For administrators only.")
        k = 0
        print("Customer name list and balances mentioned below : ")
        print("\n")
        adkey = input("Please enter admin key")
        if adkey == "12345":
            while k <= len(custName) - 1:
                print("->.Customer =", custName[k])
                print("->.Balance =", custBal[k], end=" ")
                print("-/Rs")
                print("\n")
                k = k + 1
        else:
            print("Admin Key incorrect")
        mainMenu = input("Press any key to return to the main menu")
    elif choice == "5":
        print("Thanks for accessing AAA Bank's Application")
        print("\n")
        break
    else:
        print("Invalid option! Choose between 1-5 only")
        mainMenu = input("Press any key to return to the main menu")
```

```
->.Customer = Arnav Sinha
->.Balance = 1000 -/Rs


Press any key to return to the main menu1
          ----AAA Bank----
************************************
***             MENU             ***
*** 1. New Account                ***
*** 2. Withdraw                   ***
*** 3. Deposit                    ***
*** 4. Balance                    ***
*** 5. Exit                       ***
************************************
Please enter your choice : 5
Thanks for accessing AAA Bank's Application
```

Conclusion:

Using python we created a bank management system that allowed for addition of an account, withdrawl of amounts, deposit of amounts, checking the balances and finally exiting the system. The results for each of these functions has been shown above. We can clearly see that automating this makes it extremely easy for recording and retrieving data.

END OF RECORD