MALNAD COLLEGE OF ENGINEERING, HASSAN

(An Autonomous Institution Affiliated to VTU, Belgaum)

Autonomous Programmes

Bachelor of Engineering

DEPARTMENT OF MATHEMATICS

LAB MANUAL

I Semester

DEPARTMENT OF MATHEMATICS
MCE, HASSAN

Programming in Python

Sl. No	List of Programmes (SEM-1)	CO	PO	LEVEL
1.	Expressing the function of one variable using Taylor's & Maclaurin's series.	5	1,2,5	4
2.	Finding angle between polar curves & computing the Radius of curvature of a given curve.	5	1,2,5	4
3.	Finding partial derivatives, Jacobians.	5	1,2,5	4
4.	Expressing the function of two variables using Taylor's & Maclaurin's series.	5	1,2,5	4
5.	Computation of roots using - bisection method, Newton Raphson method.	5	1,2,5	4
6.	Interpolation by- Newton's forward & Lagrange's interpolation formula.	5	1,2,5	4
7.	Numerical integration- line integral (Trapezoidal rule, Weddle's rule)	5	1,2,5	4
8.	Numerical integration- lineintegral (Simpson's 1/3rd rule, Simpson's 3/8th rule)	5	1,2,5	4
9.	Computing area by line integral & double Integral(CV/M/EE) / binomial & Poisson probability distribution (CS)	5	1,2,5	4
10.	To compute the extreme values of a function of two variables.	5	1,2,5	4

Course outcome of Mathematical procedures using python programming.

CO 5 - At the end of course, students will be able to write the program in python for the mathematical procedures connected with calculus, numerical methods, differential equations, vector calculus and execute the same with correct output.

CO	PO 1	PO 2	PO 5
CO 5	3	2	1

Rubrics for Evaluation

Daily Evaluation (for 15 Marks)	Marks	CO	PO	Level
Manual Solving	2	CO 5	PO 1	L 3
Record writing & Observation	3	CO 5	PO 1	L 3
Executing the Programme with correct output	4	CO 5	PO 2, PO 5	L 4
Final CIE	5	CO 5	PO 2, PO 5	L 4

In [1]:
```
a=3
b=4.888
c='Hi'
print(c)
```

```
Hi
```

In [2]:
```
a+b
```

Out[2]:
```
7.888
```

In [3]:
```
type(a)
```

Out[3]:
```
int
```

In [4]:
```
type(c)
```

Out[4]:
```
str
```

In [5]:
```
type(b)
```

Out[5]:
```
float
```

In [6]:
```
a="hello"
type(a)
```

Out[6]:
```
str
```

In [7]:
```
a=3
b=4.888
c='Hi'
type(a)
type(b)
```

Out[7]:
```
float
```

In [8]:
```
a=3
b=4.888
c='Hi'
display(type(a))
display(type(b))
```

```
int

float
```

Basic operators using Python

- Addition - ’+’
- Subtraction - ’-’
- Multiplication - ’*’
- Division - ’/’
- Floor Division - ’//’ Floor division operator divides the first number by the second number and rounds off the result to the nearest integer.
- Modulo - ’%’ Modulo operator divides the first number by the second number and the result is the remainder.
- Exponential - ’**’

In [9]:
```
x=19
y=100
print("x=", x)
print("The value of y is", y)
print(x,  ","      ,y)
display(x+y)
```

```
x= 19
The value of y is 100
19 , 100

119
```

In [10]:
```
# Example:Assigning numbers to variables and printing
x=9
y=2.7
print(x+y,  ','  ,x*y,  ','   ,x/y,x**y)
print(x-y)
print(x*y)
print(x,x/y,x*y)
print(x)
```

```
11.7 , 24.3 , 3.3333333333333333 377.09847446143255
6.3
24.3
9 3.3333333333333333 24.3
9
```

In [11]:
```
x=900
print(x+y)
```

```
902.7
```

In [12]:
```
a=3
b=2
c=a//b
print(a/b)
print(c)
```

```
1.5
1
```

In [13]:
```
a=23**8
b=23.**8
print(a)
print(b)
display(type(b))
display(type(a))
```

```
78310985281
78310985281.0

float

int
```

In [14]:
```
a=float(input("Enter the first number"))
b=float(input("Enter the second number"))
```

```
Enter the first number5
Enter the second number2.54
```

In [15]:
```
a=float(input("Enter the first number"))
b=float(input("Enter the second number"))
c=a+b
print("The sum of a and b", c)
print("The sum of %5.2f and %8.3f is %5.3f" %(a,b,c))
print("The sum of %5.2f and %8.3f is %5.3f", (a,b,c))
```

```
Enter the first number2.5427447
Enter the second number2.65477
The sum of a and b 5.1975147
The sum of  2.54 and    2.655 is 5.198
The sum of %5.2f and %8.3f is %5.3f (2.5427447, 2.65477, 5.1975147)
```

In [16]:
```
ab='Hello'
cd='Bye'
ab+cd
```

Out[16]:
```
'HelloBye'
```

In [17]:
```
ab*8
```

Out[17]:
```
'HelloHelloHelloHelloHelloHelloHelloHello'
```

In [18]:
```
A='Orange'
B='Shake'
D='587548'
R=234830
print(A,R)
print(B,D)
print(A+B)
print(843257+R)
```

```
Orange 234830
Shake 587548
OrangeShake
1078087
```

In [19]:
```
a=-333;b=255
print(round(a/b,3))
```

```
-1.306
```

In [20]:
```
# Assignment operators(+=,*=,%=,-=,**=,//=)
a=2
b=5
c=3
X=6
a+=2   #Equivalemt to a= a+2
b*=3   #Equivalemt to b= b*3
c**=2  #Equivalemt to c= c**2
X+=3
print(a,b,c,X)
a+=2   #Equivalemt to a= a+2
b*=3   #Equivalemt to b= b*3
c**=2  #Equivalemt to c= c**2
X+=3
print(a,b,c,X)
```

```
4 15 9 9
6 45 81 12
```

In [21]:
```
# Example:
# Write a program to find the area and perimeter of a circle of radius 'r'
r=float(input("Radius of a circle"))
p=22/7
a=p*r**2
b=2*p*r
print("area of the circle and perimter is %1.4f & %.4f" %(a,b))
```

```
Radius of a circle5
area of the circle and perimter is 78.5714 & 31.4286
```

In [1]:
```
def f(x,y,z):
    return x+y+z
f(2,4,3)
```

Out[1]: 9

In [2]:
```
sin(3.14)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_9064\1670861323.py in <module>
----> 1 sin(3.14)

NameError: name 'sin' is not defined
```

In [3]:
```
import math
math.sin(3.14)
```

Out[3]: 0.0015926529164868282

In [4]:
```
import math as m
m.acos(.5)
```

Out[4]: 1.0471975511965979

In [5]:
```
from math import *
sin(pi)
log(67)
```

Out[5]: 4.204692619390966

In [6]:
```
from math import *
from sympy import *
x=symbols('x')
y=sin(x)-x
print(y)
```

```
-x + sin(x)
```

In [7]:
```
from math import *
from sympy import *
x=symbols('x')
y=sin(x)-x
diff(y,x)
```

Out[7]: $\cos(x) - 1$

In [8]:
```
from math import *
from sympy import *
x=symbols('x')
y=sin(x)-x
diff(y,x,2)
```

Out[8]: $-\sin(x)$

In [9]:
```
from math import *
from sympy import *
x,y=symbols('x,y')
u=exp(x)*(x*cos(y)-y*sin(y))
print("u=",u)
display(u)
display(diff(u,x,x))
display(diff(u,y,y))
```

```
u= (x*cos(y) - y*sin(y))*exp(x)
```

$(x\cos(y) - y\sin(y))\,e^{x}$

$(x\cos(y) - y\sin(y) + 2\cos(y))\,e^{x}$

$-(x\cos(y) - y\sin(y) + 2\cos(y))\,e^{x}$

In [10]:
```
from sympy import *
A=Matrix([[1,2],[3,4]])
display(A)
det(A)
display(det(A))
```

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

-2

In [11]:
```
a=int(input('enter an integer:'))
b=int(input('enter an integer:'))
if a>b:
    print('a is greater than b')
else:
    print('b is greater than a')
```

```
enter an integer:-12
enter an integer:-15
a is greater than b
```

In [12]:
```
a=int(input('enter an integer:'))
if a>0:
     print('entered value is positive')
else:
     print('entered value is negative')
```

```
enter an integer:0
entered value is negative
```

In [13]:
```
a=int(input('enter an integer:'))
if a>0:
     print('entered value is positive')
elif a<0:
     print('entered value is negative')
else:
     print("number is 0")
```

```
enter an integer:0
number is 0
```

In [14]:
```
a=int(input('enter an interger:'))
b=1
while b<=10:
    print(a*b)
    b+=1
```

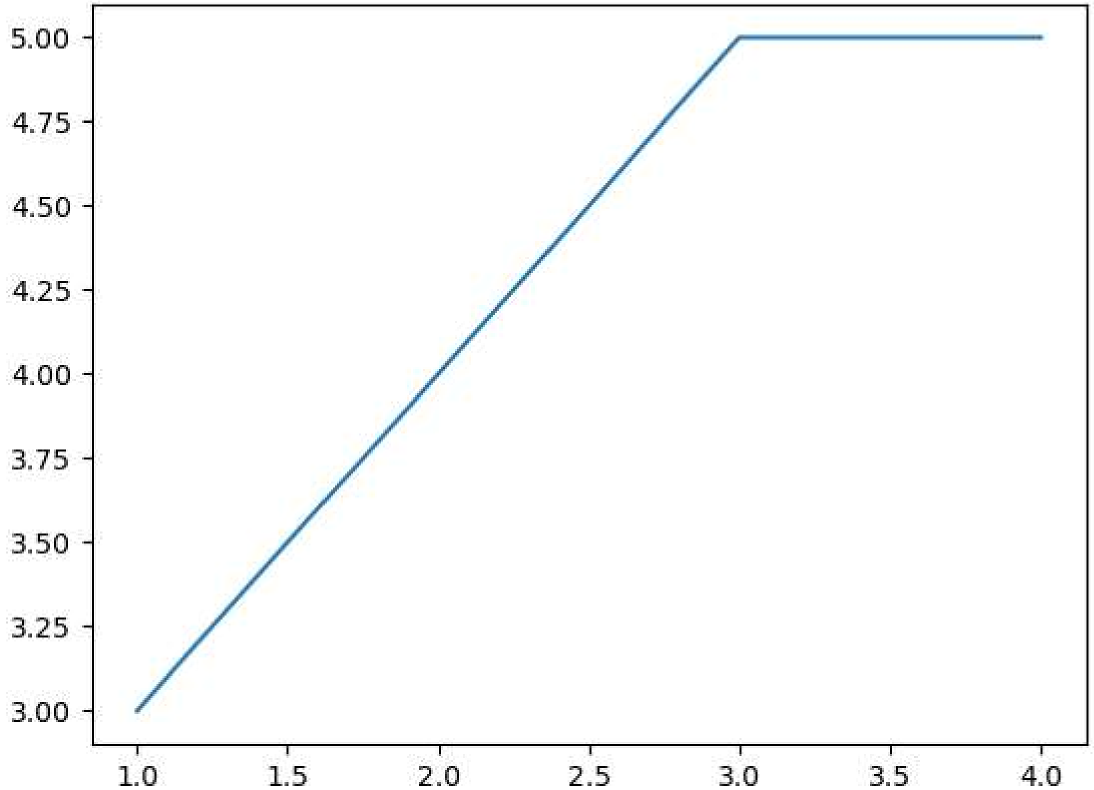
```
enter an interger:12
12
24
36
48
60
72
84
96
108
120
```

In [15]:
```
a=int(input('enter an interger:'))
for b in range(1,10,3):
    print(a*b)
```

```
enter an interger:12
12
48
84
```

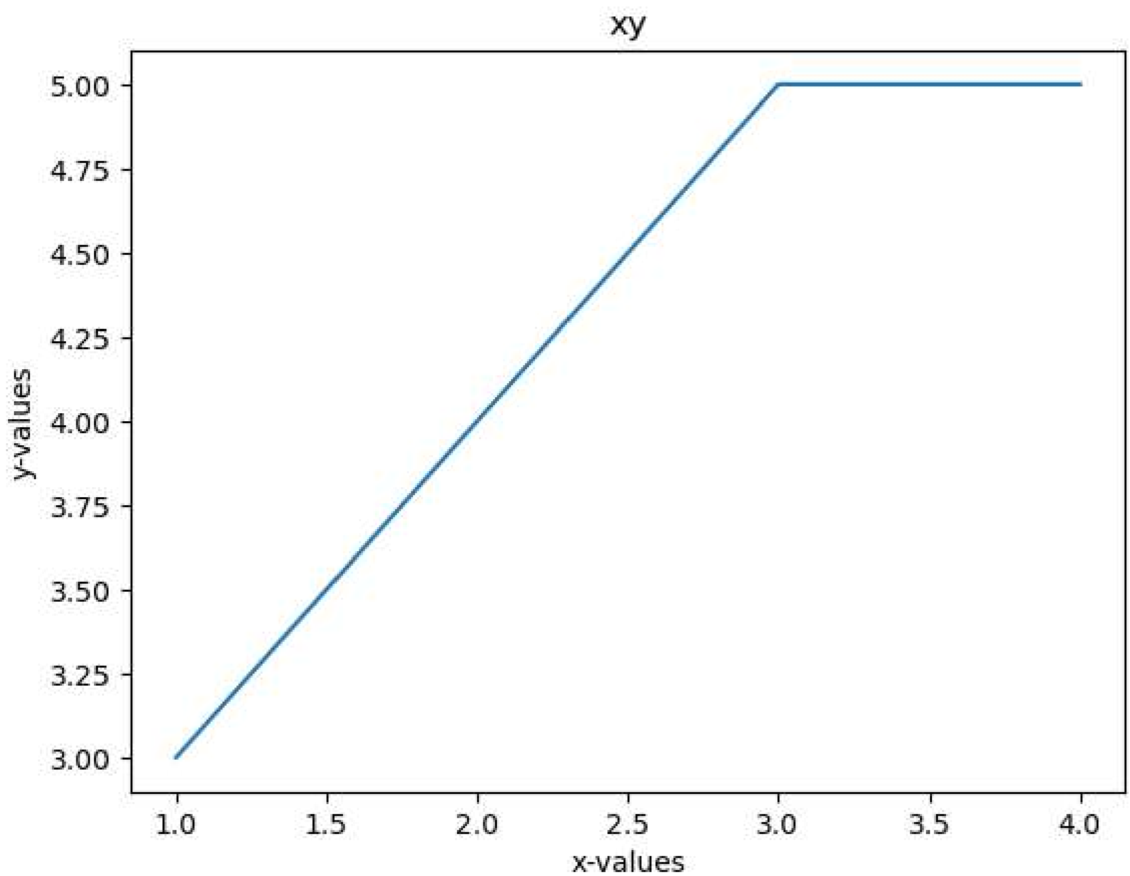
In [16]:
```
import matplotlib.pyplot as plt
x=[1,2,3,4]
y=[3,4,5,5]
plt.plot(x,y)
```

Out[16]:
```
[<matplotlib.lines.Line2D at 0x26a55c1aa90>]
```

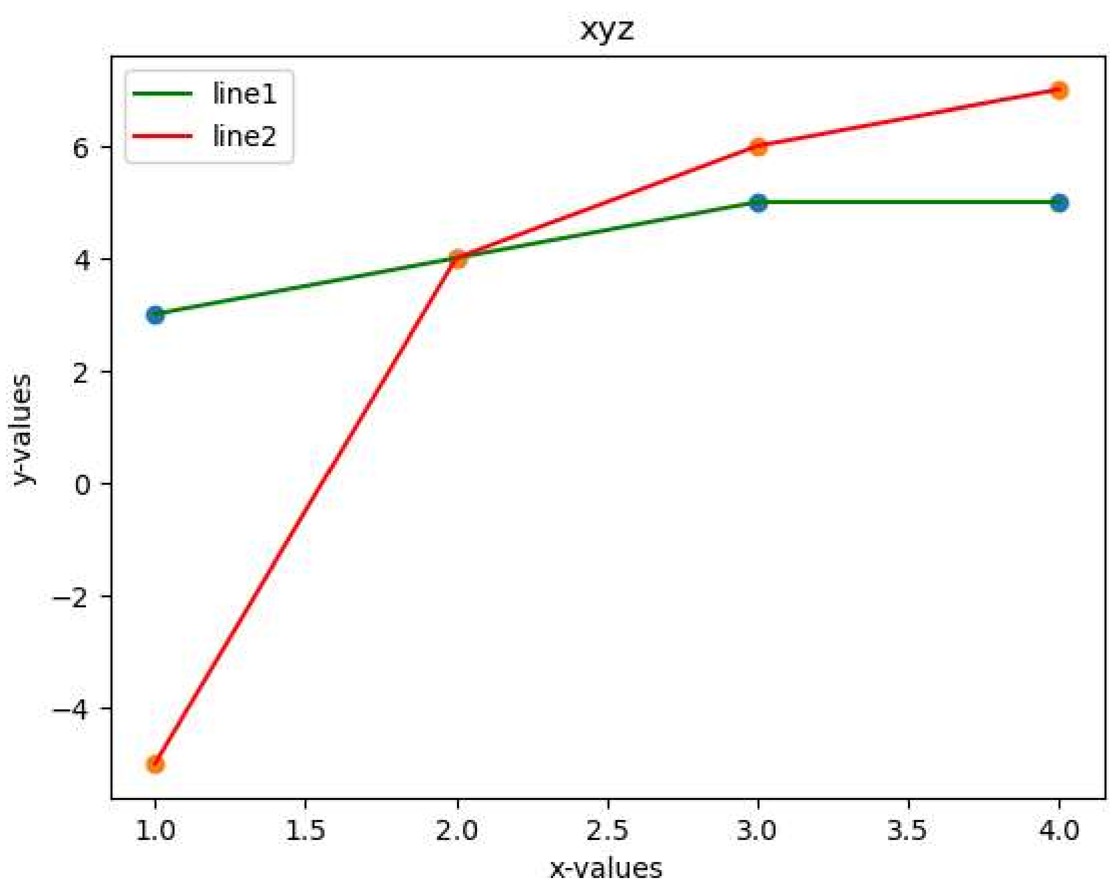


In [17]:
```
from matplotlib.pyplot import *
x=[1,2,3,4]
y=[3,4,5,5]
plot(x,y)
title('xy')
xlabel('x-values')
ylabel('y-values')
```

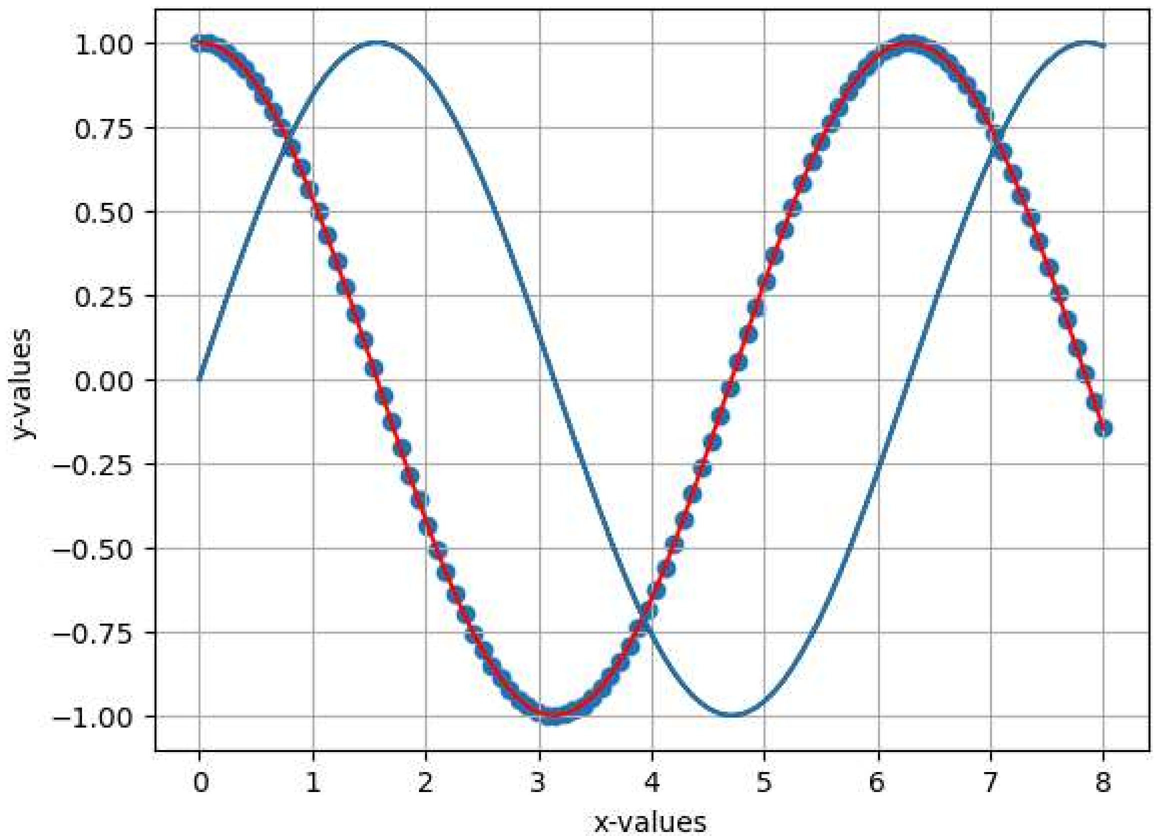
Out[17]: Text(0, 0.5, 'y-values')



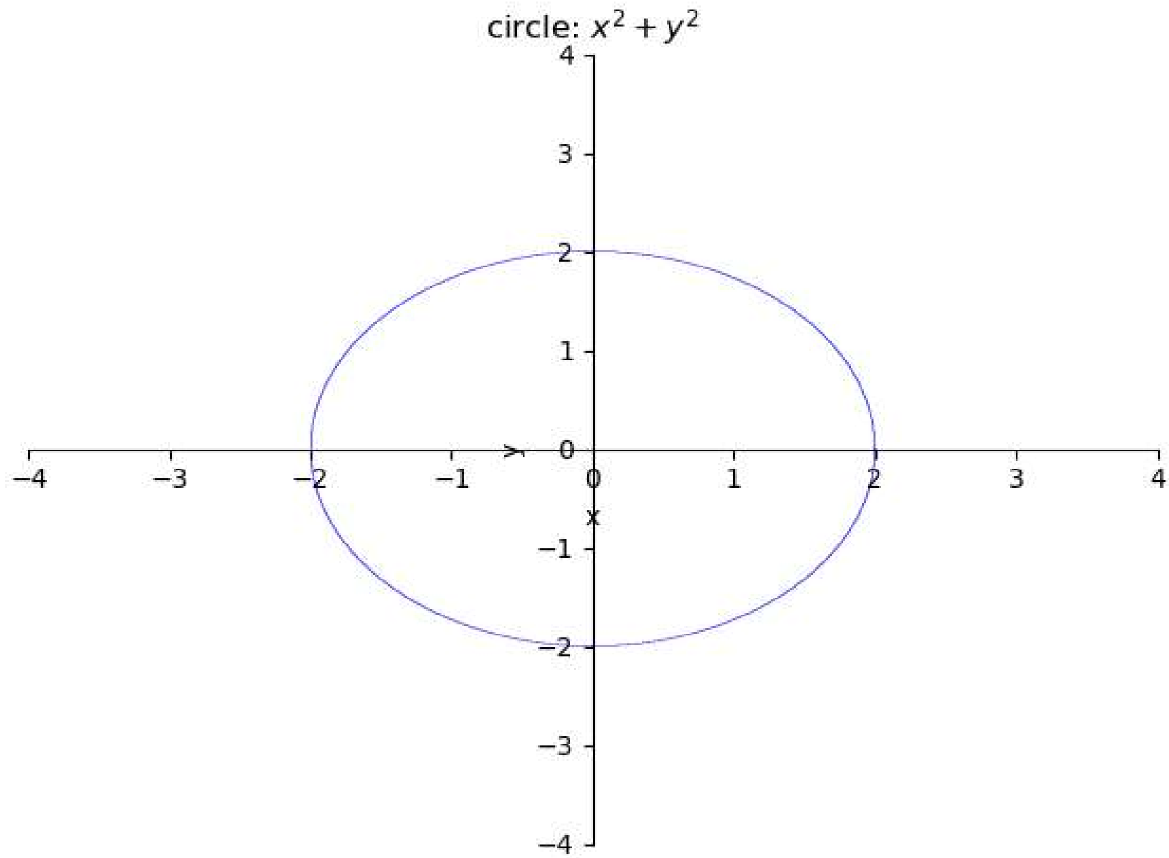
In [18]:
```
from matplotlib.pyplot import *
x=[1,2,3,4]
y1=[3,4,5,5]
y2=[-5,4,6,7]
plot(x,y1,color='green',label='line1')
plot(x,y2,color='red',label='line2')
title('xyz')
xlabel('x-values')
ylabel('y-values')
scatter(x,y1)
scatter(x,y2)
legend()
show()
```



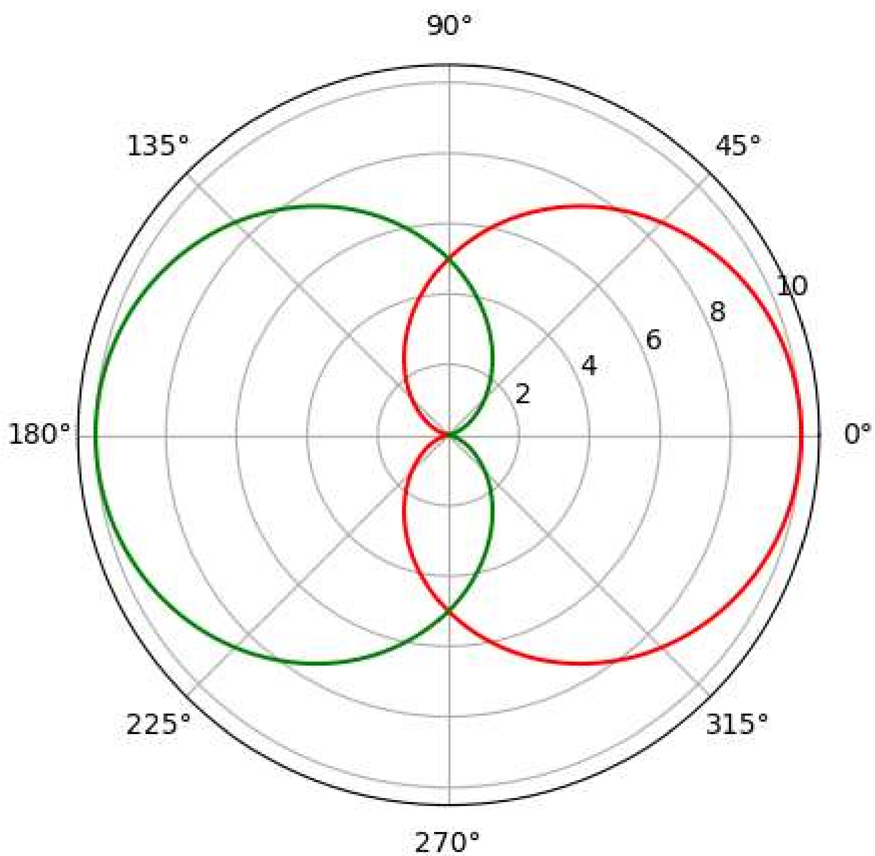
In [19]:
```
from matplotlib.pyplot import *
from numpy import *
x=linspace(0,8,100)
grid()
y1=sin(x)
y2=cos(x)
plot(x,y1,color='black',label='line1')
plot(x,y2,color='red',label='line2')
plot(x,y1)
scatter(x,y2)
xlabel('x-values')
ylabel('y-values')
show()
```



In [20]:
```
from sympy import *
x ,y = symbols('x y')
p1=plot_implicit(Eq(x**2 + y**2, 4),(x,-4,4),(y,-4,4),title= 'circle: $x^2+y^2
```



In [21]:
```
from pylab import *
theta=linspace(0,2*pi,1000)
r1=5+5*cos(theta)
polar(theta,r1,'r')
r2=5*(1-cos(theta))
polar(theta,r2,'g')
show()
```

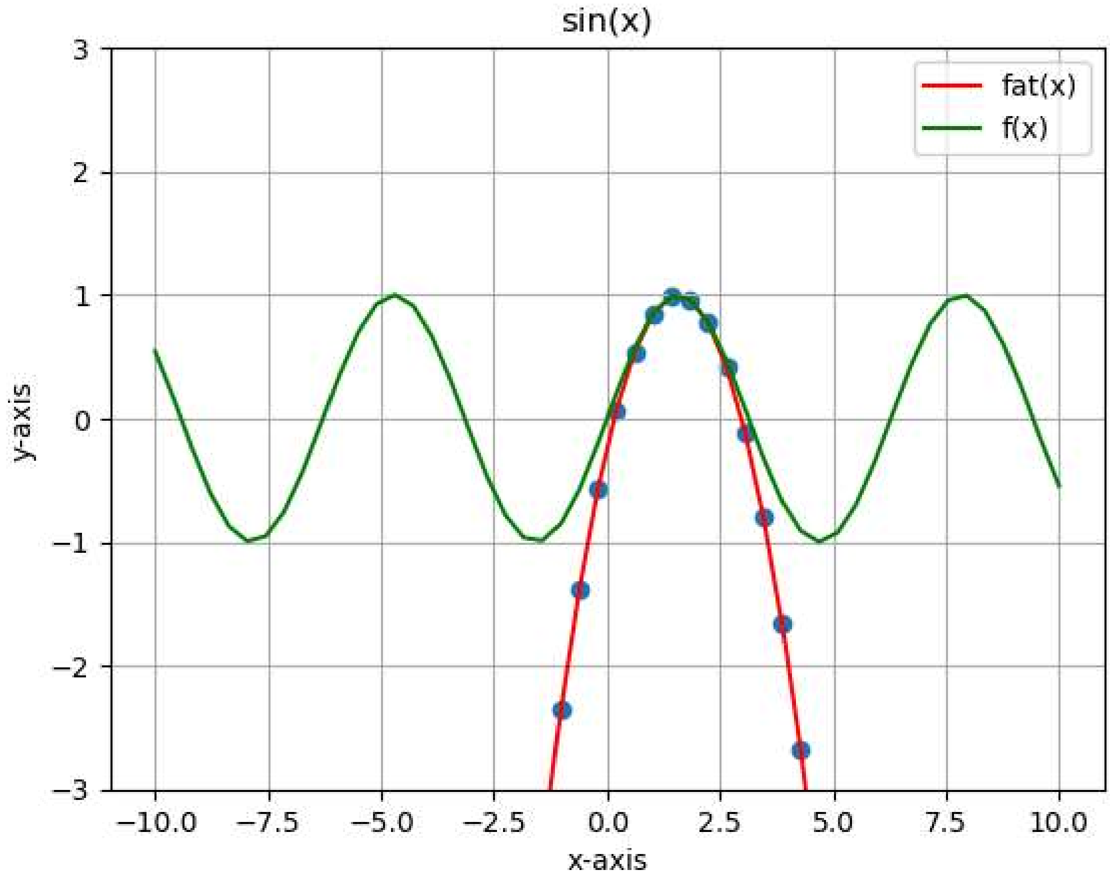


Expand sin(x) as Taylor series about x = pi/2 upto 3rd degree term. ¶

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from sympy import *
x=symbols('x')
f=sin(x)
a= float(pi/2)
df= diff(f , x)
d2f = diff(f ,x , 2)
d3f = diff(f ,x , 3)
fat = lambdify(x , f)
dfat = lambdify(x , df)
d2fat = lambdify(x , d2f)
d3fat = lambdify(x , d3f)
f=fat(a)+((x-a)/factorial(1))*dfat(a)+((x-a)**2/factorial(2))*d2fat(a)+((x-a)*
display(simplify(f))
fat = lambdify(x,f)

def f(x):
    return np.sin(x)
x=np.linspace(-10,10)
plt.plot(x, fat(x), color='red',label='fat(x)')
plt.plot(x, f(x), color='green',label='f(x)')
plt.ylim([-3 , 3])
plt.title('sin(x)')
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.scatter(x, fat(x))
plt.legend()
plt.grid()
plt.show()
```

$$-1.02053899928946 \cdot 10^{-17} x^3 - 0.5x^2 + 1.5707963267949x - 0.23370055013617$$

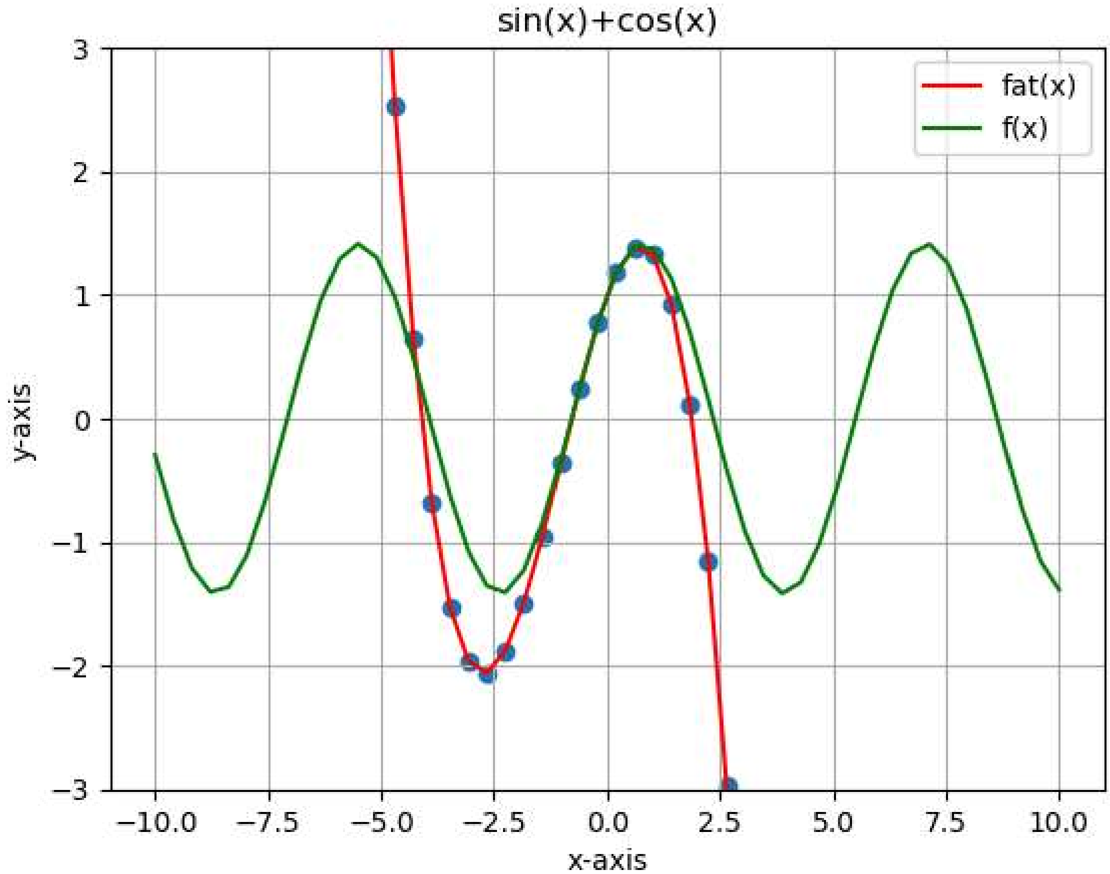


Find the Maclaurin series expansion of sin(x)+ cos(x) upto 3rd degree term.

In [2]:
```
import numpy as np
import matplotlib.pyplot as plt
from sympy import *
x=symbols('x')
f=sin(x)+cos(x)
a=0
df= diff(f , x)
d2f = diff(f ,x , 2)
d3f = diff(f ,x , 3)
fat = lambdify(x , f)
dfat = lambdify(x , df)
d2fat = lambdify(x , d2f)
d3fat = lambdify(x , d3f )
f=fat(a)+((x-a)/factorial(1))*dfat(a)+((x-a)**2/factorial(2))*d2fat(a)+((x-a)*
display(simplify(f))
fat = lambdify(x,f)

def f(x):
    return np.sin (x)+np.cos(x)
x=np.linspace(-10,10)
plt.plot(x, fat(x), color='red',label='fat(x)')
plt.plot(x, f(x), color='green',label='f(x)')
plt.ylim([-3 , 3])
plt.title('sin(x)+cos(x)')
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.scatter(x, fat(x))
plt.legend()
plt.grid()
plt.show()
```

$$-0.166666666666667x^3 - 0.5x^2 + 1.0x + 1.0$$


sin(x)+cos(x)
fat(x)
f(x)
y-axis
x-axis
3
2
1
0
−1
−2
−3
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

Find the radius of curvature $alog(sec(x/a))$

In [7]:
```
import numpy as np
from sympy import *
x, a = symbols('x, a')
y=a*log(sec(x/a))
dy=simplify(diff(y,x))
d2y=simplify(diff(y,x,2))
r=((1+dy**2)**(3/2))/d2y
print('the radius of curvature is',r)
display(r)
```

```
the radius of curvature is a*(tan(x/a)**2 + 1)**1.5*cos(x/a)**2
```

$$a\left(\tan^2\left(\frac{x}{a}\right)+1\right)^{1.5}\cos^2\left(\frac{x}{a}\right)$$

```
Finding the angle between the radius vector and the tangent:  R=a(1+cost) at
t=pi/3
```

In [8]:
```
from sympy import *
a,t=symbols('a,t')
R=a*(1+cos(t))
dRdt=diff(R,t)
R=R.subs(t,pi/3)
dRdt=dRdt.subs(t,pi/3)
PHI=atan(R/dRdt)
if PHI<0:
    PHI=PHI+pi
print('The angle between the radius vector and the tangent =',PHI)
display(PHI)
```

```
The angle between the radius vector and the tangent = 2*pi/3
```

$$\frac{2\pi}{3}$$

Find the angle between the curves r=a*(1-cost) and r=2a*(cost) at t=acos(1/3)

In [9]:
```
from sympy import *
a,t=symbols('a,t')
R=a*(1-cos(t))
dRdt=diff(R,t)
R=R.subs(t,acos(1/3))
dRdt=dRdt.subs(t,acos(1/3))
PHI=atan(R/dRdt)
if PHI<0:
    PHI=PHI+pi
r=2*a*cos(t)
drdt=diff(r,t)
r=r.subs(t,acos(1/3))
drdt=drdt.subs(t,acos(1/3))
phi=atan(r/drdt)
if phi<0:
    phi=phi+pi
print('The angle of intersection =',abs(PHI-phi))
display(abs(PHI-phi))
```

```
The angle of intersection = -0.955316618124509 + pi
```

$$-0.955316618124509 + \pi$$

If u = xy/z, v = yz/x, w = zx/y then prove that J = 4

In [1]:

```
from sympy import *
x ,y , z= symbols('x,y,z')
u=x*y/z
v=y*z/x
w=z*x/y
# find the all first order partial derivates
dux = diff(u , x)
duy = diff(u , y)
duz = diff(u , z)
dvx = diff(v , x)
dvy = diff(v , y)
dvz = diff(v , z)
dwx = diff(w , x)
dwy = diff(w , y)
dwz = diff(w , z)
# construct the Jacobian matrix
J= Matrix([[dux , duy , duz],[dvx , dvy , dvz],[dwx , dwy , dwz]])
print("The Jacobian matrix is")
display( J )
# Find the determinat of Jacobian Matrix
Jac=det( J )
print('J = ', Jac )
```

```
The Jacobian matrix is
```

$$\begin{bmatrix} \frac{y}{z} & \frac{x}{z} & -\frac{xy}{z^2} \\ -\frac{yz}{x^2} & \frac{z}{x} & \frac{y}{x} \\ \frac{z}{y} & -\frac{xz}{y^2} & \frac{x}{y} \end{bmatrix}$$

```
J =  4
```

If $u = x + 3y^2 - z^3$, $v = 4x^2yz$, $w = 2z^2 - xy$

In [3]:
```
from sympy import *
x ,y , z= symbols('x,y,z')
u=x+3*y**2-z**3
v=4*x**2*y*z
w=2*z**2-x*y
dux = diff(u , x)
duy = diff(u , y)
duz = diff(u , z)
dvx = diff(v , x)
dvy = diff(v , y)
dvz = diff(v , z)
dwx = diff(w , x)
dwy = diff(w , y)
dwz = diff(w , z)
J= Matrix([[dux , duy , duz],[dvx , dvy , dvz],[dwx , dwy , dwz]])
print("The Jacobian matrix is")
display(J)
Jac = det(J)
print('J =', Jac)
display(Jac)
```

```
The Jacobian matrix is
```

$$\begin{bmatrix} 1 & 6y & -3z^2 \\ 8xyz & 4x^2z & 4x^2y \\ -y & -x & 4z \end{bmatrix}$$

```
J = 4*x**3*y - 24*x**2*y**3 + 12*x**2*y*z**3 + 16*x**2*z**2 - 192*x*y**2*z**2
```

$$4x^3y - 24x^2y^3 + 12x^2yz^3 + 16x^2z^2 - 192xy^2z^2$$

Expand tan^-1(xy) as Taylor series about (1,1) upto 2nd degree term.

In [1]:
```
import numpy as np
from sympy import *
x,y=symbols('x,y')
f=atan(x*y)
a=1
b=1
dfx= diff(f , x)
dfy= diff(f , y)
dfxx = diff(f ,x , 2)
dfyy = diff(f ,y , 2)
dfxy = diff(f ,x , y)
fat = lambdify((x,y), f)
dfxat = lambdify((x,y), dfx)
dfyat = lambdify((x,y), dfy)
dfxxat = lambdify((x,y), dfxx)
dfyyat = lambdify((x,y), dfyy)
dfxyat = lambdify((x,y), dfxy)
f=fat(a,b)+((x-a)*dfxat(a,b)+(y-b)*dfyat(a,b))/factorial(1)+(((x-a)**2)*dfxxat
                                                              2*(x-a)*(y-b)*dfx
display(simplify(f))
```

$$-0.25x^2 + 1.0x - 0.25y^2 + 1.0y - 0.714601836602552$$

Expand sin(xy) upto 2nd degree term.

In [2]:
```
import numpy as np
from sympy import *
x,y=symbols('x,y')
f=sin(x*y)
a=0
b=0
dfx= diff(f , x)
dfy= diff(f , y)
dfxx = diff(f ,x , 2)
dfyy = diff(f ,y , 2)
dfxy = diff(f ,x , y)
fat = lambdify((x,y) , f)
dfxat = lambdify((x,y), dfx)
dfyat = lambdify((x,y), dfy)
dfxxat = lambdify((x,y), dfxx)
dfyyat = lambdify((x,y), dfyy)
dfxyat = lambdify((x,y), dfxy)
f=fat(a,b)+((x-a)*dfxat(a,b)+(y-b)*dfyat(a,b))/factorial(1)+(((x-a)**2)*dfxxat
display(simplify(f))
```

$$1.0xy$$

Bisection method: $f(x) = x^3 - x - 3$ upto 5 iterations

In [1]:
```
def f(x):
    return x**3-x-3
a=float(input('first intial limit='))
b=float(input('second intial limit='))
n=int(input('number of iterations='))
k=1
if f(a)*f(b)>0:
    print('bisection method fails')
else:
    while k<=n:
        xn=(a+b)/2
        if f(a)*f(xn)<0:
            b=xn
        else:
            a=xn
        print('root of the given equation',xn)
        k+=1
```

```
first intial limit=1
second intial limit=2
number of iterations=5
root of the given equation 1.5
root of the given equation 1.75
root of the given equation 1.625
root of the given equation 1.6875
root of the given equation 1.65625
```

Bisection method: $f(x) = xsin(x) - 1$ upto 5 iterations

In [2]:

```
from math import *
def f(x):
    return x*sin(x)-1
a=float(input('first intial limit='))
b=float(input('second intial limit='))
n=int(input('number of iterations='))
k=1
if f(a)*f(b)>0:
    print('bisection method fails')
else:
    while k<=n:
        xn=(a+b)/2
        if f(a)*f(xn)<0:
            b=xn
        else:
            a=xn
        print('root of the given equation',xn)
        k+=1
```

```
first intial limit=1
second intial limit=2
number of iterations=5
root of the given equation 1.5
root of the given equation 1.25
root of the given equation 1.125
root of the given equation 1.0625
root of the given equation 1.09375
```

Newton-Raphson method $x^3 - x^2 - 2$ upto 8 iterations around 1

In [3]:

```
def f(x):
    return x**3-x**2-2
def df(x):
    return 3*x*x-2*x
xo=float(input('intial value='))
n=int(input('number of iterations='))
k=1
while(k<=n):
    xn=xo-f(xo)/df(xo)
    print('root=',xn,'at iteration',k)
    xo=xn
    k+=1
```

```
intial value=1
number of iterations=8
root= 3.0 at iteration 1
root= 2.238095238095238 at iteration 2
root= 1.839867776037989 at iteration 3
root= 1.7096795196984376 at iteration 4
root= 1.6957728017350469 at iteration 5
root= 1.695620787604337 at iteration 6
root= 1.6956207695598622 at iteration 7
root= 1.695620769559862 at iteration 8
```

Newton-Raphson method: $xlog(x) - 1.2$ upto 12 iterations around 1

In [4]:
```
from math import *
def f(x):
    return x*log(x)-1.2
def df(x):
    return 1+log(x)
xo=float(input('intial value='))
n=int(input('number interations='))
k=1
while(k<=n):
    xn=xo-f(xo)/df(xo)
    print('root=',xn,'at iteration',k)
    xo=xn
    k+=1
```

```
intial value=1
number interations=12
root= 2.2 at iteration 1
root= 1.901079710006334 at iteration 2
root= 1.8881138482423665 at iteration 3
root= 1.8880867531472094 at iteration 4
root= 1.8880867530283434 at iteration 5
root= 1.8880867530283436 at iteration 6
root= 1.8880867530283434 at iteration 7
root= 1.8880867530283436 at iteration 8
root= 1.8880867530283434 at iteration 9
root= 1.8880867530283436 at iteration 10
root= 1.8880867530283434 at iteration 11
root= 1.8880867530283436 at iteration 12
```

Using Lagrange's interpolation formula find y(10) given x=[5,6,9,11] & y=[12,13,14,16]

In [1]:
```
x=list(eval(input('Enter x points:')))
y=list(eval(input('Enter y points:')))
X=float(input('given value:'))
n=len(x)
sum1=0
for i in range(0,n):
    term=1
    for j in range(0,n):
        if i!=j:
            term=term*(X-x[j])/(x[i]-x[j])
    sum1+=term*y[i]
print('\n the estimated value of y(X)=',sum1)
```

```
Enter x points:5,6,9,11
Enter y points:12,13,14,16
given value:10

 the estimated value of y(X)= 14.666666666666668
```

Trapezoidal method:

In [2]:
```
def f(x):
    return 1/(1+x**2)
a=float(input('lower limit='))
b=float(input('upper limit='))
n=int(input('Number of intervals='))
h=(b-a)/n
k=1
sum=0
while(k<n):
    xn=a+k*h
    sum=sum+f(xn)
    k=k+1
tra_f=(h/2)*(f(a)+f(b)+2*sum)
print('value of integration',tra_f)

from sympy import *
x=symbols('x')
int_f=integrate(f(x),[x,a,b])
print('exact value of integration',float(int_f))
```

```
lower limit=0
upper limit=1
Number of intervals=3
value of integration 0.7807692307692307
exact value of integration 0.7853981633974483
```

Simpson's 1/3 rule:

In [3]:

```
def f(x):
    return 1/(1+x)
a=float(input('lower limit='))
b=float(input('upper limit='))
n=int(input('Number of intervals='))
h=(b-a)/n
k=1
sum=0
while(k<n):
    xn=a+k*h
    if (k%2==0):
        sum=sum+2*f(xn)
    else:
        sum=sum+4*f(xn)
    k=k+1
simp3_f=(h/3)*(f(a)+f(b)+sum)
print('value of integration',simp3_f)

from sympy import *
x=symbols('x')
int_f=integrate(f(x),[x,a,b])
print('exact value of integration',float(int_f))
```

```
lower limit=1
upper limit=3
Number of intervals=8
value of integration 0.6931545306545306
exact value of integration 0.6931471805599453
```

Simpson's rule 3/8 rule:

In [1]:
```
def f(x):
    return 1/((1+x)**2)
a=float(input('lower limit='))
b=float(input('upper limit='))
n=int(input('Number of intervals='))
h=(b-a)/n
k=1
sum=0
while(k<n):
    xn=a+k*h
    if (k%3==0):
        sum=sum+2*f(xn)
    else:
        sum=sum+3*f(xn)
    k=k+1
simp8_f=((3*h)/8)*(f(a)+f(b)+sum)
print('value of integration',simp8_f)

from sympy import *
x=symbols('x')
int_f=integrate(f(x),[x,a,b])
print('exact value of integration',float(int_f))
```

```
lower limit=0
upper limit=3
Number of intervals=6
value of integration 0.7582621173469386
exact value of integration 0.75
```

Evaluate: ∫ [$-y\ dx$ + $x\ dy$] c along the curve C: y =x^2 from (0,0) to (1,1).

In [2]:
```
from sympy import *
x,y=symbols('x,y')
y=x**2
f=-y*diff(x,x)+x*diff(y,x)
soln=integrate(f,[x,0,1])
print("I=", soln)
display(soln)
```

```
I= 1/3
```

$$\frac{1}{3}$$

Evaluate: ∫ [$xy\ dx$ + $x^{2}z\ dy$ + $xyz\ dz$] c along the curve C: x=exp(t), y=exp(-t), z=t^2, 1 ≤ t ≤ 2

In [5]:
```
from sympy import *
x,y,z,t=symbols('x,y,z,t')
x=exp(t)
y=exp(-t)
z=t**2
f=x*y*diff(x,t)+x**2*z*diff(y,t)+x*y*z*diff(z,t)
soln=integrate(f,[t,1,2])
print("I=", soln)
display(soln)
```

```
I= 15/2 - exp(2)
```

$$\frac{15}{2} - e^2$$

$$Evaluate : \int \int x^3 e^y dydx; 0 < x < 1, 0 < y < 1$$

In [6]:
```
from sympy import *
x,y=symbols('x,y')
f=x**3*exp(y)
soln=integrate(f,[y,0,1],[x,0,1])
print("I=", soln)
display(soln)
```

```
I= -1/4 + E/4
```

$$-\frac{1}{4} + \frac{e}{4}$$

$Evaluate$:∬[x2+y2]$dydx$;0<x<1,0<y<x

In [8]:
```
from sympy import *
x,y=symbols('x,y')
f=x**2+y**2
soln=integrate(f,[y,0,x],[x,0,1])
print("I=", soln)
display(soln)
```

```
I= 1/3
```

$$\frac{1}{3}$$

when a coin is tossed 6 times, what is the probability to get a)no head,b) atmost 2 heads,c) atleast one head, d)exactly 2 heads using binomial probability distribution

In [8]:
```
from math import *
def binomial_probability(n, k, p):
    return comb(n, k) * (p**k) * ((1 - p)**(n - k))


n = 6
p = 0.5


no_heads_prob = binomial_probability(n, 0, p)

at_most_2_heads_prob = sum(binomial_probability(n, k, p) for k in range(3))

at_least_one_head_prob = 1 - no_heads_prob

exactly_2_heads_prob = binomial_probability(n, 2, p)


print(f"Probability of getting no heads: {no_heads_prob:.4f}")
print(f"Probability of getting at most 2 heads: {at_most_2_heads_prob:.4f}")
print(f"Probability of getting at least one head: {at_least_one_head_prob:.4f}
print(f"Probability of getting exactly 2 heads: {exactly_2_heads_prob:.4f}")
```

```
Probability of getting no heads: 0.0156
Probability of getting at most 2 heads: 0.3438
Probability of getting at least one head: 0.9844
Probability of getting exactly 2 heads: 0.2344
```

out of 800 families with 5 children each, find the expected number of families with a) atleast one boy,b) 2 or 3 boys, c)2 girls, d) atleast 2 boys and e)no girls

In [9]:
```
from math import *


def binomial_probability(n, k, p):
    return comb(n, k) * (p**k) * ((1 - p)**(n - k))


total_families = 800


n = 5
p = 0.5

no_boys_prob = binomial_probability(n, 0, p)
at_least_one_boy_prob = 1 - no_boys_prob


two_boys_prob = binomial_probability(n, 2, p)
three_boys_prob = binomial_probability(n, 3, p)
two_or_three_boys_prob = two_boys_prob + three_boys_prob
two_girls_prob = binomial_probability(n, 2, 1 - p)
at_least_2_boys_prob = sum(binomial_probability(n, k, p) for k in range(2, 6))
no_girls_prob = binomial_probability(n, 5, p)

expected_families_at_least_one_boy = at_least_one_boy_prob * total_families
expected_families_two_or_three_boys = two_or_three_boys_prob * total_families
expected_families_two_girls = two_girls_prob * total_families
expected_families_at_least_2_boys = at_least_2_boys_prob * total_families
expected_families_no_girls = no_girls_prob * total_families


print(f"Expected number of families with at least one boy: {expected_families_a
print(f"Expected number of families with exactly 2 or 3 boys: {expected_famili
print(f"Expected number of families with exactly 2 girls: {expected_families_tw
print(f"Expected number of families with at least 2 boys: {expected_families_at
print(f"Expected number of families with no girls (all boys): {expected_famili
```

```
Expected number of families with at least one boy: 775.00
Expected number of families with exactly 2 or 3 boys: 500.00
Expected number of families with exactly 2 girls: 250.00
Expected number of families with at least 2 boys: 650.00
Expected number of families with no girls (all boys): 25.00
```

the probability that a fuse manufactured by a company being defective is 2%. find the probability that a box containing 200 fuses have a) no defective, b)2 or more, c)atleast one defective fuses

In [10]:
```
from math import *


def poisson_probability(lam, k):
    return (lam ** k * exp(-lam)) / factorial(k)

n = 200
p = 0.02


lam = n * p
no_defective_prob = poisson_probability(lam, 0)
two_or_more_defective_prob = 1 - (poisson_probability(lam, 0) + poisson_probab
at_least_one_defective_prob = 1 - no_defective_prob

print(f"Probability of no defective fuses: {no_defective_prob:.4f}")
print(f"Probability of 2 or more defective fuses: {two_or_more_defective_prob:
print(f"Probability of at least one defective fuse: {at_least_one_defective_pro
```

```
Probability of no defective fuses: 0.0183
Probability of 2 or more defective fuses: 0.9084
Probability of at least one defective fuse: 0.9817
```