#1. From the data in the following table, find the value of P(50) by using Lagrange Interpolation.

```
x = [0, 20, 40, 60, 80, 100]
y = [26.0, 48.6, 61.6, 71.2, 74.8, 75.2]

m = len(x)
n = m - 1   #degree of the polynomial

xp = float(input("Enter x: "))
yp = 0
for i in range(n + 1):
    p = 1
    for j in range(n + 1):
        if j != i:
            p *= (xp - x[j])/(x[i] - x[j])
    yp += y[i]*p

print("For x = %2f, y = %f" %(xp, yp))
```

Output

```
Enter x: 50
For x = 50.000000, y = 66.947656
```

```
#2. Bisection Method
def f(x):
    y = x**3 - 4*x - 9
    return y

a = int(input("Enter the value of a:"))
b = int(input("Enter the value of b:"))
c = (a+b)/2

while abs(f(c)) > 0.01:
    if f(a)*f(c) > 0:
        a = c
    else:
        b = c

    c = (a+b)/2

    print(c, " ", f(c))

print("The Root of x**3 -4*x - 9 is", c)
```

Output

```
Enter the value of a:2
Enter the value of b:3
2.75   0.796875
2.625   -1.412109375
2.6875   -0.339111328125
2.71875   0.220916748046875
2.703125   -0.061077117919921875
2.7109375   0.07942342758178711
2.70703125   0.009049236774444458
The Root of x**3 -4x - 9 is 2.70703125
```

```
#3. Newton-Raphson Method
#f(x)=2*x**3-2*x-5 and the root lies between 1 and 2
def f(x):
    return 2*x**3-2*x-5
def df(x):
    return 6*x**2-2

a=float(input("Initial Value: "))
n=int(input("Number of Iterations: "))
k=1
while(k<=n):
    r=a-(f(a)/df(a))
    print("Root is",r,"at",k,"iteration")
    k=k+1
    a=r
```

Output

```
Initial Value: 1.5
Number of Iterations: 4
Root is 1.608695652173913 at 1 iteration
Root is 1.6006452475271589 at 2 iteration
Root is 1.6005985465000654 at 3 iteration
Root is 1.6005985449336209 at 4 iteration
```

```
#4. Regula-Falsi Method

def f(x):
    return x**3-x-2

a = int(input("First Initial Guess:"))
b = int(input("Second Initial Guess:"))
n = int(input("Number of Interactions:"))

if f(a)>f(b):
    print("Regula-Falsi Method Fails")
else:
    k = 1
    while(k<=n):
        c = (a*f(b)-b*f(a))/(f(b)-f(a))
        if f(a)*f(c)<0:
            b = c
        else:
            a = c
        print("Root is",c,"at",k,"iteration")
        k=k+1
```

Output

```
First Initial Guess:1
Second Initial Guess:2
Number of Interactions:5
Root is 1.3333333333333333 at 1 iteration
Root is 1.4626865671641789 at 2 iteration
Root is 1.504019003949949 at 3 iteration
Root is 1.5163305647602632 at 4 iteration
Root is 1.5199185500233559 at 5 iteration
```

```
#5. Trapezoidal Rule of Integration
def f(x):
    return 1/(1+x*x)
a = float(input("Lower Limit:"))
b = float(input("Upper Limit:"))
n = int(input("Number of Strips:"))
h = (b - a)/n
k = 1
sum = 0
while(k<n):
    t = a + k * h
    sum = sum + f(t)
    k = k + 1
int_a = (h/2)*(f(a) + f(b) + 2*sum)
print("Value of Integration:",int_a)

import sympy as sy
x = sy.Symbol("x")
int_e = sy.integrate(f(x), (x, 0, 1))
print("Exact Value:",int_e)
```

Output

```
Lower Limit:0
Upper Limit:1
Number of Strips:4
Value of Integration: 0.7827941176470589
Exact Value: pi/4
```

```
#6. Simpson's 1/3 rule of Integration
def f(x):
        return 1/(1+x)

a=float(input("Enter the Lower Limit:"))
b=float(input("Enter the Lower Limit:"))
n=int(input("Enter the number of strips:"))

h=(b-a)/n
k=1
sum=0

while(k<n):
    x=a+k*h
    if(k%2==0):
        sum=sum+2*f(x)
    else:
        sum=sum+4*f(x)
    k=k+1
Ia=(h/3)*(f(a)+f(b)+sum)
print("Approx value of Integration:",Ia)
```

Output

```
Enter the Lower Limit:1
Enter the Lower Limit:3
Enter the number of strips:8
Approx value of Integration: 0.6931545306545306
```

```
#7. Simpson's 3/8 rule of Integration

import math
def f(x):
   return 1/math.sqrt(1+x)

a=float(input("Lower Limit:"))
b=float(input("Upper Limit:"))
n=int(input("Enter the value of n:")
# n should be a multiple of 3
h=(b-a)/n

k=1
sum=0
while(k<n):
    x=a+k*h
    if(k%3==0):
        sum=sum+2*f(x)
    else:
        sum=sum+3*f(x)
    k=k+1
Ia=((3*h)/8*(f(a)+f(b)+sum))
print("Approx value of Integration:",Ia)
```

Output

```
Lower Limit:0
Upper Limit:6
Enter the value of n:6
Approx value of Integration: 3.2991454807609544
```

```
#8. Gauss Elimination method of solving simultaneous equations
import numpy as np
def gausselim(a,b):
    n=len(b)
    for k in range(0,n-1):
        for i in range (k+1,n):
            if a[i,k] != 0.0:
                l = a[i,k]/a[k,k]
                a[i,k+1:n] = a[i,k+1:n]-l*a[k,k+1:n]
                b[i] = b[i]-l*b[k]
    for k in range (n-1,-1,-1):
        b[k] = (b[k]-np.dot(a[k,k+1:n],b[k+1:n]))/a[k,k]
    return b

a = np.array([[4,-2,1],[-2,4,-2],[1,-2,4]])
b = np.array([11,-16,17])
print("The Solution is",gausselim(a,b))
```

Output

```
The Solution is [ 1 -1  5]
```

```
#9. Gauss-Siedal Method of solving simultaneous equations
# 2x+y+z=5
# 3x+5y+2z=15
# 2x+y+4z=8

x=0
y=0
z=0
for i in range(7):
    print("Iteration",i+1)
    x=(5-y-z)/2
    y=(15-3*x-2*z)/15
    z=(8-2*x-y)/4

    print(x,y,z)
    print()
```

Output

```
Iteration 1
2.5 0.5 0.625

Iteration 2
1.9375 0.5291666666666667 0.8989583333333333

Iteration 3
1.7859375 0.5229513888888889 0.9762934027777777

Iteration 4
1.7503776041666668 0.5197520254629631 0.9948731915509259

Iteration 5
1.7426873914930556 0.5188127628279321 0.9989531135464891

Iteration 6
1.7411170618127896 0.5185828391645769 0.999795759302461

Iteration 7
1.7408107007664808 0.5185317586063757 0.9999617099651655
```

#10. RK fourth order method of solving differential equations

```
def f(x,y):
    return (x+y)**2
x = 0
y = 1
xmax = 0.2 # y(xmax) will be calculated
stepsize = 0.05 # Size by which x values should differ
while x<xmax:
    k1 = stepsize * f(x,y)
    k2 = stepsize * f(x+stepsize/2, y + k1/2)
    k3 = stepsize * f(x+stepsize/2, y + k2/2)
    k4 = stepsize * f(x + stepsize, y+k3)
    deltay = (k1+2*k2+2*k3+k4)/6
    y = y + deltay
    x = x + stepsize
    print(x,y)
```

Output

```
0.05 1.055355603267246
0.1 1.123048901982997
0.15000000000000002 1.2060878670291046
0.2 1.308497619150346
```