```
Newton interpolation

import numpy as np
import matplotlib.pyplot as plt

def coef(x, y):
   '''x : array of data points
      y : array of f(x)  '''
   x.astype(float)
   y.astype(float)
   n = len(x)
   a = []
   for i in range(n):
       a.append(y[i])

   for j in range(1, n):

       for i in range(n-1, j-1, -1):
           a[i] = float(a[i]-a[i-1])/float(x[i]-x[i-j])

   return np.array(a) # return an array of coefficient
"""''' a : array returned by function coef()
       x : array of data points
       r : the node to interpolate at  '''"""
def Eval(a, x, r):
   x.astype(float)
   n = len( a ) - 1
   temp = a[n]
   for i in range( n - 1, -1, -1 ):
       temp = temp * ( r - x[i] ) + a[i]
   return temp # return the y_value interpolation
```

Lagrange interpolation

```
def lagrange(X,Y,x,):
   try:


       Total = 0
       for i in range(0, len(X)):
           numerator = 1
           denominator = 1
           fx=0
           for b in range(0, len(X)):
               if i != b:
                   numerator *= (x-X[b])
                   denominator *= (X[i]-X[b])
           fx = Y[i] * (numerator/denominator)
           print(fx)
           Total += fx
       return Total
   except ZeroDivisionError:
```

```python
        print("division by zero")


X = [-2, -1, 0, 4]
Y = [-2, 4, 1, 8]

print(lagrange(X,Y,2))
```

Golden search method

```python
import math
import matplotlib.pyplot as plt
import numpy as np
phi = (math.sqrt(5)-1)/2

def goldensearch(f,a,b,precision):
   error =100
   while error > precision :
        d = phi*(b-a)
        x1 = a + d
        x2 = b - d
       # error = (abs(f(x1)-f(x2)))
        if f(x1)<f(x2):
            a = x2
        else:
            f(x1)>f(x2)
            b = x1

        error -= 10
   return (x1+x2)/2

f = lambda x: x**2
def g(x):
   return x**2- 6*x + 15
x_max =(goldensearch(f,-4,4,0.01))
max = f(x_max)
print("max =", max)
x = np.linspace(-4,4,1000)
y = f(x)
plt.plot(x,y,label ='golden search',color='green')
plt.plot(x_max, max,'bo',
              label='max[f(x)]', markersize=5, markeredgewidth=1)
plt.grid(linestyle='-', linewidth=0.25)
plt.legend()
plt.show()
```

Newton rhapson
```
def dx(f, x):
   return abs(0-f(x))

def newtons_method(f, df, x0, e):
   delta = dx(f, x0)
   while delta > e:
       x0 = x0 - f(x0)/df(x0)
       delta = dx(f, x0)
   print ('Root is at: ', x0)
   print ('f(x) at root is: ', f(x0))

def f(x):
   return 6*x**5-5*x**4-4*x**3+3*x**2

def df(x):
   return 30*x**4-20*x**3-12*x**2+6*x


print(newtons_method(f,df,0.1,0.001))
```

Bisection

```
def bisection(f,a,b,N):
   if f(a)*f(b) >= 0:
       print("Bisection method fails.")
       return None
   first = a
   last = b
   for n in range(1,N+1):
       midpoin = (first + last)/2
       f_midpoin = f(midpoin)
       if f(first)*f_midpoin < 0:
           first = first
           last = midpoin
       elif f(last)*f_midpoin < 0:
           first = midpoin
           last = last
       elif f_midpoin == 0:
           print("Found exact solution.")
           return midpoin
       else:
           print("Bisection method fails.")
           return None
   return (first + last)/2

f = lambda x: x**2 - x - 1
print(bisection(f,1,2,25))
 #  1.618033990263939
#f = lambda x: (2*x - 1)*(x - 3)
```

```
#bisection(f,0,1,10)
# 0.5
```

Numerical difff

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

def forward(f,x,h):
   return (f(x+h)-f(x))/h

def backward(f,x,h):
   return (f(x)- f(x-h))/h

def central(f,x,h):
   return (f(x+h)-f(x-h))/h

def f(x):
   return x**2

print("forward= ",forward(f,2,0.1) )
print("backward= ",backward(f,2,0.1) )
print("central= ",central(f,2,0.1) )

x = np.linspace(-5,5,10)
plt.plot(x,forward(f,x,0.1),label="forward")
plt.plot(x,backward(f,x,0.1),label="backward")
plt.plot(x,central(f,x,0.1),label="central")
plt.legend()
plt.show()
```