```python
import numpy as np
#The values a and b are the endpoints, and m is the number of quadrature
#points (minus one because we start from zero)
def CompTrapezium(m,a,b):
    def f(x):
        return np.sin(np.exp(x)) #Just defining the specific function
    h =(b-a)/m #The definition of h
    j = 0
    while j <= m: #Setting up a while loop to create list objects
        if j == 0:#That store the data points. Need to establish 1st case
            x_list = [a]
            f_list = [f(a)]
        else:
            x_j = a + j*(b-a)/m #establishing the general case
            x_list.append(x_j)
            f_list.append(f(x_j))
        j+=1
    sum = 0.0 #Setting up a sum as a counter for each iteration
    for k in np.arange(m+1):
        if k == 0 or k == m:
            sum += f_list[k]/2.0 #Be careful of the endpoints!
        else:
            sum += f_list[k] #The interior points need no coefficient
    sum = h*sum
    print sum

CompTrapezium(40,0.0,2.0)  #Testing it all out. The numbers are the
In  = 0.560960063979       #found values for h = 1/10, 1/20, 1/40
I2n = 0.553291387475
I4n = 0.55151561097
R   = (In-I2n)/(I2n-I4n)
print R

def CompSimpson(m,a,b): #Definition of args same as comp trapezoidal rule
    def f(x):
        return np.sin(np.exp(x))
    h = (b-a)/m
    s = f(a)+f(b) #Setting up the endpoints
    for j in np.arange(1,m,2): #Establishing the 4 coeffs for the even case
        s += 4*f(a+j*h)
    for j in np.arange(2,m-1,2): #Establishing the 2 coeffs for the odd case
        s += 2*f(a+j*h)
    print s * h/3

CompSimpson(400,0.0,2.0) #evaluation using h = 1/10,1/20, and 1/40
In  = 0.546748419238
I2n = 0.550735161974
I4n = 0.550923685469
R   = (In-I2n)/(I2n-I4n) #R=21.1471930117
print R
In = 0.550934886678      #evaluation for h = 1/100,1/200,1/400. Notice the
I2n = 0.55093515586      #Better approximation to our guess of R for larger
I4n = 0.550935172623     #N (R = 16.0581041708 here)
R   = (In-I2n)/(I2n-I4n)
print R
```