```python
import sys
import math
import sympy as sp
import numpy as np
from tabulate import tabulate

#calculate the nth derivative of fx
#returns a list of [fx, fx', fx'', ... , fx^(n)]
def n_deriv(fx, n):
   x = sp.symbols('x')
   f = fx
   deriv_list = [f]
   for i in range(1, n + 1):
         df_i = deriv_list[-1].diff(x).replace(sp.Derivative, lambda *args: f(x))
         deriv_list.append(df_i)
   return deriv_list

#takes a function, bounds, and step size and finds the x value where the function is maximized
def find_max_error(f_n, bounds, step):
   max_y = None
   max_x = None
   x = sp.symbols('x')
   for i in np.arange(bounds[0], bounds[1] + step, step):
      y = abs(f_n.subs(x, i))
      if (max_y is None and max_x is None) or y > max_y:
         max_y = y
         max_x = i
   return max_x

#takes parameters h, n, f, a, b, and a boolean that checks if we want Simpson's method or not
#returns the calculated numerical integral of f
def numerical_integral(h, n, f, a, b, simpsons):
   x = sp.symbols('x')
   integral = 0

   #iterate through all points
   for i in range(n):
      #calculate xi by taking the start point + step size * current index
      xi = a + i*h
      xi = round(xi, 10)

      #checks to see if we want to use Simpson's method or not
      if simpsons:
         #i==0 or i==n-1 corresponds to the two end points
         if i == 0 or i == n - 1:
            integral += f.subs(x, xi)
         #multiply the even terms by 2
         elif i%2 == 0:
            integral += 2 * f.subs(x, xi)
         #multiply the odd terms by 4
         else:
            integral += 4 * f.subs(x, xi)
      #if not using Simpson's, we are using Trapezoidal method
      else:
         #check for end points
         if i == 0 or i == n - 1:
            integral += f.subs(x, xi)
         #multiply non-end points by 2
         else:
            integral += 2 * f.subs(x, xi)

   if simpsons:
      integral *= h/3
   else:
      integral *= h/2
```

```
    return integral

"""
defining our initial variables
"""
x = sp.symbols('x')

#define our function 1/(1+e^(-3x))
f = sp.sympify(1/(1+sp.exp(-3*x)))

tolerance = float(input("Enter max tolerance: "))
a = float(input("a = "))
b = float(input("b = "))

real = sp.integrate(f, (x, a, b))

"""
finds where the second and fourth derivatives are maximized
used for calculating our h-values
"""
second_deriv = n_deriv(f, 2)[-1]
fourth_deriv = n_deriv(f, 4)[-1]

second_deriv_max = find_max_error(second_deriv, [a, b], 0.01)
second_deriv_max = abs(second_deriv.subs(x, second_deriv_max))

fourth_deriv_max = find_max_error(fourth_deriv, [a, b], 0.01)
fourth_deriv_max = abs(fourth_deriv.subs(x, fourth_deriv_max))

"""
calculate h-values, interals, and points using the Composite error terms
"""
h_t = ((tolerance*12)/((b-a)*second_deriv_max))**(1/2)
h_s = ((tolerance*180)/((b-a)*fourth_deriv_max))**(1/4)

interval_t = math.ceil((b-a)/h_t)

#simpsons needs an even interval (resulting in odd # of points)
interval_s = math.ceil((b-a)/h_s)
if interval_s%2 != 0:
    interval_s += 1

points_t = interval_t+1
points_s = interval_s+1

#check to make sure the math we did above is actually correct and in tolerance
assert second_deriv_max*((b-a)/points_t)**2*((b-a)/12) <= tolerance
assert fourth_deriv_max*((b-a)/points_s)**4*((b-a)/180) <= tolerance


"""
setting up our method of calculating each integral
"""
points = [points_t, points_s]
simps = [False, True]
h_vals = [h_t, h_s]

#for loop that uses either the trapezoid method or simpsons method because I didn't want to copy/paste
for calc_type in range(len(points)):
    print("\n=-=-=-=-=-=\nComposite {0}:\n=-=-=-=-=-=".format("Trapezoidal" if calc_type == 0 else "Simpson's"))
    print("Required h = {0}".format(h_vals[calc_type]))
    print("=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=")

    #sets up our n values
    n_vals = [11, 41, points[calc_type]]
    err_vals = [0 for _ in range(len(n_vals))]
```

```python
data = [[] for _ in range(len(n_vals))]

#iterates through n values and calculates integral with each n
for i in range(len(n_vals)):
    n = n_vals[i]

    #calculate our h-value for this specific n
    h = ((b-a)/(n-1))

    integral = numerical_integral(h, n, f, a, b, simps[calc_type])

    #calculates absolute and relative error
    abs_err = abs(real - integral)
    rel_err = abs(abs_err/real)*100

    data_row = [n, h, real, integral, abs_err, rel_err, abs_err <= tolerance]
    data[i] = data_row

#prints out table of information
print(tabulate(data, headers=["n",
                              "h",
                              "Real Value",
                              "Calculated Value",
                              "Absolute Error",
                              "Relative Error",
                              "Within Tolerance"]))

print("\n=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=\n")
```