```
import numpy as np

def func(eq, t0, tx, y0, h):
    n = int((tx-t0)/h)
    t = [t0]
    y = [y0]

    for i in range (0,n,1):
        k1 = eq(t[i], y[i])
        k2 = eq(t[i]+h, y[i] * h+k1)
        yn1 = y[i] + (h/2)*(k1 + k2)
        tn1 = t[i] + h
        y.append(yn1)
        t.append(tn1)
        print('%d\t%.2f\t%.4f'% (i+1,tn1,yn1) )
        print('----------------------')
    return (t, y) #tuple containing two arrays (t,y)

def eq(t, y):
    return np.exp(t)-y

ans = func(eq, 0, 5, 10, 0.5)
```

```
1       0.50    9.1622
----------------------
2       1.00    8.6965
----------------------
3       1.50    8.7298
----------------------
4       2.00    9.4859
----------------------
5       2.50    11.3458
----------------------
6       3.00    14.9489
----------------------
7       3.50    21.3592
----------------------
8       4.00    32.3388
----------------------
9       4.50    50.8007
----------------------
10      5.00    81.5539
----------------------
```

```
def euler_method(f, t_initial, t_final, y_initial, h):
    """
    Solves the ordinary differential equation y' = f(t, y) using Euler's method w
    Returns arrays of time and y values at each step.
    """
    num_steps = int((t_final - t_initial) / h)
    t_values = [t_initial]
    y_values = [y_initial]
    print('\n-------SOLUTION-------')
    print('----------------------')
    print('#\ttn\tyn')
```

```
    print('---------------------')
    for i in range(num_steps):
        slope = f(t_values[i], y_values[i])
        y_new = y_values[i] + h * slope
        t_new = t_values[i] + h
        y_values.append(y_new)
        t_values.append(t_new)
        print('%d\t%.2f\t%.4f'% (i+1,t_new,y_new) )
        print('---------------------')
    return t_values,y_values


def RK4_method(f, t_initial, t_final, y_initial, h):
    """
    Solves the ordinary differential equation y' = f(t, y) using Euler's method with step
    Returns arrays of time and y values at each step.
    """
    num_steps = int((t_final - t_initial) / h)
    t_values = [t_initial]
    y_values = [y_initial]
    print('\n-------SOLUTION-------')
    print('---------------------')
    print('#\ttn\tyn')
    print('---------------------')
    for i in range(num_steps):
        k1 = f(t_values[i], y_values[i])
        k2 = f(t_values[i]+0.5*h, y_values[i]+0.5*k1)
        k3 = f(t_values[i]+h/2, y_values[i]+k2/2)
        k4 = f(t_values[i]+h, y_values[i]+k3)
        y_new = y_values[i] + (h/6) * (k1+2*k2+2*k3+k4)
        t_new = t_values[i] + h
        y_values.append(y_new)
        t_values.append(t_new)
        print('%d\t%.2f\t%.4f'% (i+1,t_new,y_new) )
        print('---------------------')
    return t_values,y_values



from tabulate import tabulate
import numpy as np
from scipy.integrate import solve_ivp
import numpy as np

def compareFunc(f, t0, tx, y0, h):
  n = int((tx - t0) / h)
  data=[]

  sol = solve_ivp(f, [0, 1], [1], t_eval=np.array(np.linspace(t0+h, tx, n)))
  ya=sol.y.flatten()
  yl=np.insert(ya,0,y0)
  te,ye=euler_method(f, t0, tx, y0, h)
  th,yh=func(f, t0, tx, y0, h)
  trk4,yrk4=RK4_method(f, t0, tx, y0, h)
```

```
    error1 = abs(yl - ye)
    error2 = abs(yl - yh)
    error3 = abs(yl - yrk4)

    print('\n\t\t\tERRORS')
    print('-------------------------------------------------------------')
    print('n\tStep\tEuler\t\tHeun\t\tRK4')
    print('-------------------------------------------------------------')
    for i in range(len(te)):
        print('%d\t%.2f\t%.6f\t%.6f\t%.6f' % (i+1, te[i], error1[i], error2[i], error3[i])
        print('-------------------------------------------------------------')

    if(error1.all()<error2.all() and error1.all()<error3.all()):
      print("Method with least error is Euler")
    elif(error2.all()<error1.all() and error2.all()<error3.all()):
      print("Method with least error is Hueuns")
    else:
      print("Method with least error is RK4")

compareFunc(eq, 0, 1, 1, 0.5)
```

```
-------SOLUTION-------
---------------------
#       tn      yn
---------------------
1       0.50    1.0000
---------------------
2       1.00    1.3244
---------------------
1       0.50    1.2872
---------------------
2       1.00    1.8059
---------------------

-------SOLUTION-------
---------------------
#       tn      yn
---------------------
1       0.50    1.1132
---------------------
2       1.00    1.4673
---------------------

                        ERRORS
-------------------------------------------------------------
n       Step    Euler           Heun            RK4
-------------------------------------------------------------
1       0.00    0.000000        0.000000        0.000000
-------------------------------------------------------------
2       0.50    0.127213        0.159968        0.013981
-------------------------------------------------------------
3       1.00    0.218965        0.262528        0.076065
-------------------------------------------------------------
Method with least error is RK4
```

```
def eq1(t, y):
```

```
    return y**2-t-1
def eq2(t, y):
    return y-t
def eq3(t, y):
    return (t-1)**2-y

compareFunc(eq1, 0, 1, 1, 0.25)
compareFunc(eq2, 0, 1, 1, 0.25)
compareFunc(eq3, 0, 1, 1, 0.25)
```

```
-------SOLUTION-------
----------------------
#       tn      yn
----------------------
1       0.25    1.0000
----------------------
2       0.50    0.9375
----------------------
3       0.75    0.7822
----------------------
4       1.00    0.4977
----------------------
1       0.25    0.8516
----------------------
2       0.50    0.6106
----------------------
3       0.75    0.3697
----------------------
4       1.00    0.2072
----------------------

-------SOLUTION-------
----------------------
#       tn      yn
----------------------
1       0.25    0.9407
----------------------
2       0.50    0.7069
----------------------
3       0.75    0.3602
----------------------
4       1.00    0.0017
----------------------

                        ERRORS
--------------------------------------------------------------
n       Step    Euler           Heun            RK4
--------------------------------------------------------------
1       0.00    0.000000        0.000000        0.000000
--------------------------------------------------------------
2       0.25    0.036566        0.111872        0.022760
--------------------------------------------------------------
3       0.50    0.113942        0.212937        0.116680
--------------------------------------------------------------
4       0.75    0.243228        0.169316        0.178816
--------------------------------------------------------------
5       1.00    0.395960        0.105427        0.099998
--------------------------------------------------------------
```

```
Method with least error is RK4

-------SOLUTION-------
----------------------
#       tn      yn
----------------------
1       0.25    1.2500
----------------------
2       0.50    1.5000
```

Colab paid products - Cancel contracts here

✓ 0s completed at 11:15 PM