```python
# -*- coding: utf-8 -*-
"""
Created on Mon Nov  4 14:18:51 2019

@author: Alexm
"""

import numpy as np
import matplotlib.pyplot as plt
print ("Alexandra Mulholland 17336557")
#assingment 4
#solving and analysing numerical solution to ordinary differential equation
#(1 + t)x + 1 − 3t + t^2
#must plot direction field (x versus t)
#consists of arrows of gradient dx/dt (given by ODE)

#PART 1
#producing said direction field

def f(T,X):
    return ((1+T)*X)+1-(3*T)+T**2
#defining ranges of grid axes
xmin, xmax = -3, 3
tmin, tmax = 0, 5

#defining number of steps
dimpts=25
x=np.linspace(xmin, xmax, dimpts)
t=np.linspace(tmin, tmax, dimpts)
T, X = np.meshgrid(t, x)



#25x25 grid points
#arrows are tangent to the solutions
plt.figure(1)
plt.quiver(T,X,1,f(T,X), scale=50)
plt.ylabel('x(t)')
plt.xlabel('t')
plt.title('Direction field for dx/dt')
plt.ylim(-3.0,3.0)
plt.xlim(0,5.0)
plt.show()

#PART 2

#Defining approximations for the ODE
#I have not produced an analytical solution as i think this ODE is unsolvable
#not linear nor separable
#defining the simple euler
def seuler(T,X,step):
    X_new = X + step*f(T,X)
    return X_new

#PART 3

#improved euler method
def ieuler(T,X,step):
 X_new = X + 0.5*step*(f(T,X) + f(T + step,X+step*f(T,X)))
 return X_new

#runge kutta method
def rk(T,X,step):
 k1 = f(T,X)
 k2 = f(T+ 0.5*step,X + 0.5*step*k1)
 k3 = f(T + 0.5*step,X + 0.5*step*k2)
 k4 = f(T + step,X + step*k3,)
 X_new = X + step/6.0*(k1 + 2.0*k2 + 2.0*k3 + k4)
 return X_new

#initial step size =0.04
#then reduced in part 3 to 0.02, called step1
```

```
#initial condition x(t=0)=x0=0.0655
step = 0.04
start = 0
end = 5
X_zero = 0.0655
t_zero= 0
step1=0.02

n = int((end-start)/step)
t1 = np.arange(start,end,step)

seul = np.zeros(n)
ieul = np.zeros(n)
ruku = np.zeros(n)

seul[0] = X_zero
ieul[0] = X_zero
ruku[0] = X_zero

for i in range(1,n):
    seul[i] = seuler(t1[i-1], seul[i-1], step)
    ieul[i] = ieuler(t1[i-1], ieul[i-1], step)
    ruku[i] = rk(t1[i-1], ruku[i-1], step)

#PART 3b-reducing step size

seul1 = np.zeros(n)
ieul1 = np.zeros(n)
ruku1 = np.zeros(n)

seul1[0] = X_zero
ieul1[0] = X_zero
ruku1[0] = X_zero

for i in range(1,n):
    seul1[i] = seuler(t1[i-1], seul1[i-1], step1)
    ieul1[i] = ieuler(t1[i-1], ieul1[i-1], step1)
    ruku1[i] = rk(t1[i-1], ruku1[i-1], step1)




plt.figure(2)
plt.quiver(T, X, 1, f(T,X),scale=50)
plt.plot(t1,seul,color='red')
plt.ylim(-3,3)
plt.title('Euler method atop direction field for step size=0.04')
plt.ylabel("x")
plt.xlabel("t")
plt.show()

plt.figure(3)
plt.quiver(T, X, 1, f(T,X),scale=50)
plt.plot(t1,seul,color='red',label='Euler method')
plt.plot(t1,ieul,color='green',label='Improved Euler method')
plt.plot(t1,ruku,color='blue',label='Runge-Kutta method')
plt.ylim(-3,3)
plt.title('Plot of the three numerical methods atop the direction field,step=0.04')
plt.ylabel("x")
plt.xlabel("t")
plt.legend()
plt.show()


#PART 3b- step size halved




plt.figure(8)
plt.quiver(T, X, 1, f(T,X),scale=50)
plt.plot(t1,seul1,color='orange',label='Euler method')
```

```
plt.plot(t1,ieul1,color='brown',label='Improved Euler method')
plt.plot(t1,ruku1,color='purple',label='Runge-Kutta method')
plt.ylim(-3,3)
plt.title('Plot of the three numerical methods atop the direction field,step=0.02')
plt.ylabel("x")
plt.xlabel("t")
plt.legend()
plt.show()


plt.figure(6)
plt.quiver(T, X, 1, f(T,X),scale=50)
plt.plot(t1,seul1,color='orange',label='step size 0.02')
plt.ylim(-3,3)
plt.title('Euler method atop direction field for step size=0.02 and 0.04')
plt.plot(t1,seul,color='red',label='step size 0.04')
plt.legend(loc='upper right')
plt.ylabel("x")
plt.xlabel("t")
plt.show()


plt.quiver(T, X, 1, f(T,X), scale=50)
plt.plot(t1,ieul,color='green', label='step size 0.04')
plt.plot(t1,ieul1,color='brown', label='step size 0.02')
plt.ylim(-3,3)
plt.title('Improved Euler method atop direction field for step size=0.02 and 0.04')
plt.ylabel("x")
plt.xlabel("t")
plt.legend(loc='upper right')
plt.show()

plt.figure()
plt.quiver(T, X, 1, f(T,X), scale=50)
plt.plot(t1,ruku,color='blue', label='step size 0.04')
plt.plot(t1,ruku1,color='purple', label='step size 0.02')
plt.ylim(-3,3)
plt.title('Runge-Kutta method for step size=0.02 and 0.04')
plt.ylabel("x")
plt.xlabel("t")
plt.legend(loc='upper right')
plt.show()
```