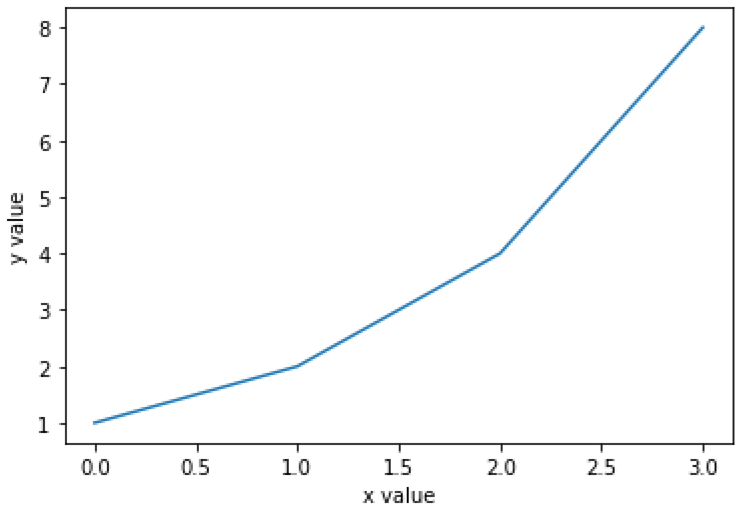
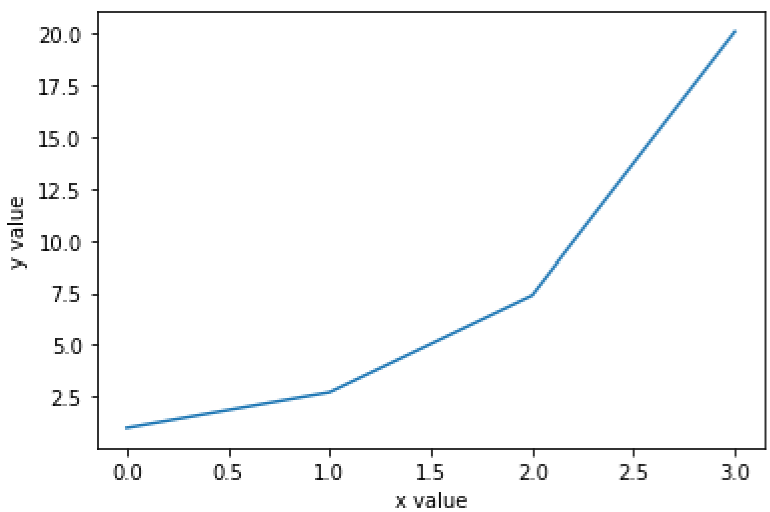
Question 1

```
# derivative function f
def f(y):
  return y

# euler formula function
def euler(x0, y, h, x):
  x_list = []
  y_list = []
  while x0 < x:
    x_list.append(x0)
    y_list.append(y)
    y = y + h * f(y)
    x0 = x0 + h

  return x_list, y_list, y

# initial values and input x
x0, y0, h, x = 0, 1, 1, 4

x_list, y_list, y = euler(x0, y0, h, x)
print("Approximate of y({}) using euler's method is: {}".format(x, y))

# plotting the graph
plt.plot(x_list, y_list)
plt.xlabel("x value")
plt.ylabel("y value")
plt.show()
```

```
# actual graph : y = e^x
y_list1 = np.exp(x_list)
plt.plot(x_list, y_list1)
plt.xlabel("x value")
plt.ylabel("y value")
plt.show()
```

```
# output below
Approximate of y(4) using euler's method is: 16
```



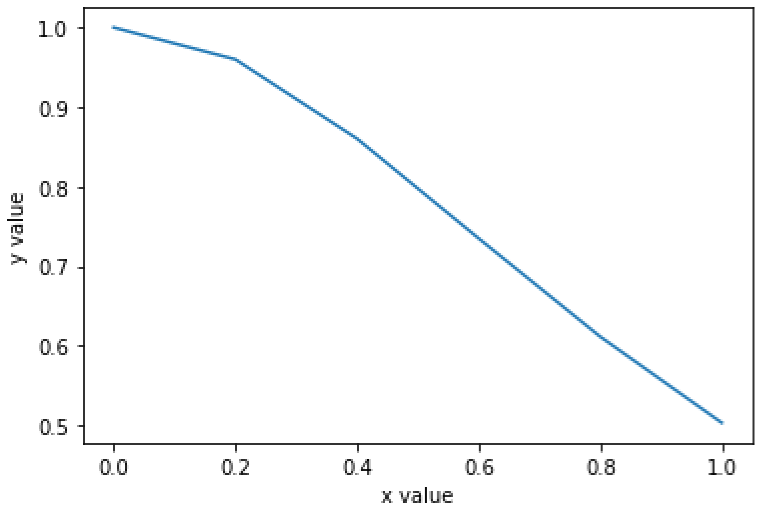


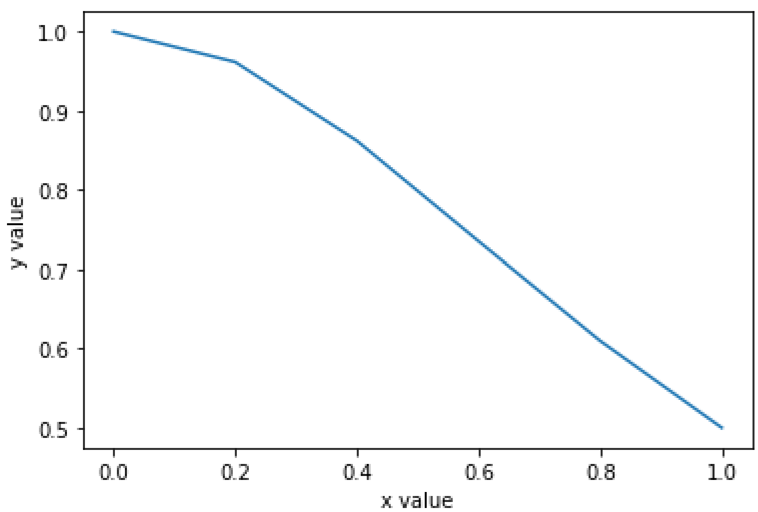
Question 2

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

# derivative function
def f(x,y):
  return -2*x*y**2

# modified euler method function
def modified_euler(x0, y0, xi, h, n):
  # setting up the x and y arrays which will contain the x,y values
  x = np.linspace(x0,xi,n)
```

```
  y = np.zeros([n])
  y[0] = y0
  p1 = np.zeros([n])
  p1[0] = 0

  for i in range(1,n):
    p1[i] = h*f(x[i-1],y[i-1]) + y[i-1]
    y[i] = h/2*(f(x[i],p1[i]) + f(x[i-1],y[i-1])) + y[i-1]

  return y

# initial values and input x (xi)
x0, y0, xi, h = 0, 1, 1, 0.2
n = int((xi - x0)/h) + 1
x = np.linspace(x0,xi,n)
y = modified_euler(x0, y0, xi, h, n)
print("Aprroximamte value of y({}) using modified euler's method is: {}".format(xi, y[n-1]))

# plotting the graph
plt.plot(x,y)
plt.xlabel("x value")
plt.ylabel("y value")
plt.show()

y = 1/(1+x**2)
# actual graph: y=1/1-x^2
plt.plot(x,y)
plt.xlabel("x value")
plt.ylabel("y value")
plt.show()
# output below
```

```
Aprroximamte value of y(1) using modified euler's method is:
0.503338255442106
```





Question 3

```
from math import e
import numpy as np

# derivative function
def f(t, y):
    return 3*e**(-t) + 0.4*y

# Runge-Kutta method function
def runge_kutta(x0, y0, h):
  # setting array t = [0,3]
  t = np.arange(0,3.1,h)
```

```
  y = y0
  for i in range(0, np.size(t)):
    # calculating k1,k2,k3,k4
    k1 = h * f(t[i], y)
    k2 = h * f(t[i] + 0.5 * h, y + 0.5 * k1)
    k3 = h * f(t[i] + 0.5 * h, y + 0.5 * k2)
    k4 = h * f(t[i] + h, y + k3)
    y = y + (1.0 / 6.0)*(k1 + 2 * k2 + 2 * k3 + k4)
    print(i+1, round(t[i], 1), round(k1, 3), round(k2, 3), round(k3, 3), round(k4, 3), round(y, 3),
sep='\t')

# input values
x0 = 0
y = 5
h = 0.1
print("i\tt[i]\tk1\tk2\tk3\tk4\ty")
runge_kutta(x0, y, h)
# output below
```

i	t[i]	k1	k2	k3	k4	y
1	0.0	0.5	0.495	0.495	0.491	5.495
2	0.1	0.491	0.488	0.488	0.485	5.983
3	0.2	0.485	0.483	0.483	0.481	6.466
4	0.3	0.481	0.48	0.48	0.479	6.946
5	0.4	0.479	0.479	0.479	0.479	7.425
6	0.5	0.479	0.48	0.48	0.481	7.904
7	0.6	0.481	0.482	0.482	0.484	8.387
8	0.7	0.484	0.487	0.487	0.49	8.874
9	0.8	0.49	0.493	0.493	0.497	9.367
10	0.9	0.497	0.501	0.501	0.505	9.868
11	1.0	0.505	0.51	0.51	0.515	10.377
12	1.1	0.515	0.52	0.52	0.526	10.898
13	1.2	0.526	0.532	0.533	0.539	11.43
14	1.3	0.539	0.546	0.546	0.553	11.976
15	1.4	0.553	0.56	0.561	0.568	12.537
16	1.5	0.568	0.577	0.577	0.585	13.114
17	1.6	0.585	0.594	0.594	0.603	13.708
18	1.7	0.603	0.613	0.613	0.622	14.32
19	1.8	0.622	0.632	0.633	0.643	14.953
20	1.9	0.643	0.654	0.654	0.665	15.607
21	2.0	0.665	0.676	0.676	0.688	16.283
22	2.1	0.688	0.7	0.7	0.713	16.983
23	2.2	0.713	0.725	0.725	0.738	17.709
24	2.3	0.738	0.752	0.752	0.766	18.461
25	2.4	0.766	0.78	0.78	0.794	19.24
26	2.5	0.794	0.809	0.809	0.824	20.05
27	2.6	0.824	0.84	0.84	0.856	20.889

28	2.7	0.856	0.872	0.872	0.889	21.762
29	2.8	0.889	0.906	0.906	0.923	22.667
30	2.9	0.923	0.941	0.941	0.959	23.608
31	3.0	0.959	0.978	0.978	0.997	24.586