```
#define function for part a
def func_1 (x,y):
    return x**2+y**2

#define partial derivative with respect to x and y for function 1
def func1deriv(x,y):
    return 2*x,2*y

#gradient descent
def grad_f1(x,y,learnrate):
    x0=0.1
    y0=0.1
    learnrate=0.1
    for i in range(10): #max iteration
        grad_x,grad_y=func1deriv(x0,y0)
        x=x0-learnrate*grad_x
        y=y0-learnrate*grad_y
    return x,y

answer_x,answer_y=grad_f1(0.1,0.1,0.1)
print(answer_x,answer_y)
```

```
0.08 0.08
```

```
#define function for part b
def func_2 (x,y):
    return x**2+y**2

#define partial derivative with respect to x and y for function 2
def func2deriv(x,y):
    return 2*x,2*y

#gradient descent
def grad_f2(x,y,learnrate):
    x0=-1
    y0=1
    learnrate=0.01
    for i in range(10): #max iteration
```

```
    learnrate=0.01
    for i in range(10): #max iteration
        grad_x,grad_y=func2deriv(x0,y0)
        x=x0-learnrate*grad_x
        y=y0-learnrate*grad_y
    return x,y

answer_x,answer_y=grad_f2(-1,1,0.01)
print(answer_x,answer_y)
```

```
-0.98 0.98
```

```
import numpy as np # for exponential function
#define function for part c
def func_3 (x,y):
    return 1-np.exp(-x**2-(y-2)**2)-2*np.exp(-x**2-(y+2)**2)

#define partial derivative with respect to x & y for function 3
def func3deriv(x,y):
    return 2*x*np.exp(-x**2-(y-2)**2)+4*x*np.exp(-x**2-(y+2)**2),2*(y-2)*np.exp(-x**2-(y-2)**2)+4*(y+2)*np.exp(-x**2-(y+2)**2)
#gradient descent
def grad_f3(x,y,learnrate):
    x0=0
    y0=1
    learnrate=0.01
    for i in range(1000): #max iteration
        grad_x,grad_y=func3deriv(x0,y0)
        x=x0-learnrate*grad_x
        y=y0-learnrate*grad_y
    return x,y

answer_x,answer_y=grad_f3(0,1,0.01)
print(answer_x,answer_y)
```

```
0.0 1.0073427796469385
```

```
import numpy as np # for exponential function
#define function for part d
def func_4 (x,y):
    return 1-np.exp(-x**2-(y-2)**2)-2*np.exp(-x**2-(y+2)**2)
```

```
import numpy as np # for exponential function
#define function for part d
def func_4 (x,y):
    return 1-np.exp(-x**2-(y-2)**2)-2*np.exp(-x**2-(y+2)**2)
#define partial derivative with respect to x & y for function 4
def func4deriv(x,y):
    return 2*x*np.exp(-x**2-(y-2)**2)+4*x*np.exp(-x**2-(y+2)**2),2*(y-2)*np.exp(-x**2-(y-2)**2)+4*(y+2)*np.exp(-x**2-(y+2)**2)
#gradient descent
def grad_f4(x,y,learnrate):
    x0=0
    y0=-1
    learnrate=0.01
    for i in range(1000): #max iteration
        grad_x,grad_y=func4deriv(x0,y0)
        x=x0-learnrate*grad_x
        y=y0-learnrate*grad_y
    return x,y

answer_x,answer_y=grad_f4(0,-1,0.01)
print(answer_x,answer_y)
```

```
0.0 -1.0147077730586125
```

```
#code greg gave us
import numpy as np
from mpl_toolkits import mplot3d #for 3D plots
import matplotlib.pyplot as plt #usual matplotlib
%matplotlib widget
X=np.linspace(-5,5,100)
Y=np.linspace(-5,5,100)
x,y=np.meshgrid(X,Y)
z= 1-np.exp(-x**2-(y-2)**2)-2*np.exp(-x**2-(y+2)**2)
4
fig = plt.figure()
ax = plt.axes(projection='3d')
ax.plot_surface(x, y, z,cmap='viridis', edgecolor='none')
#x,y z are variable names.
```

```
<mpl_toolkits.mplot3d.art3d.Poly3DCollection at 0x7f7935aab890>
```