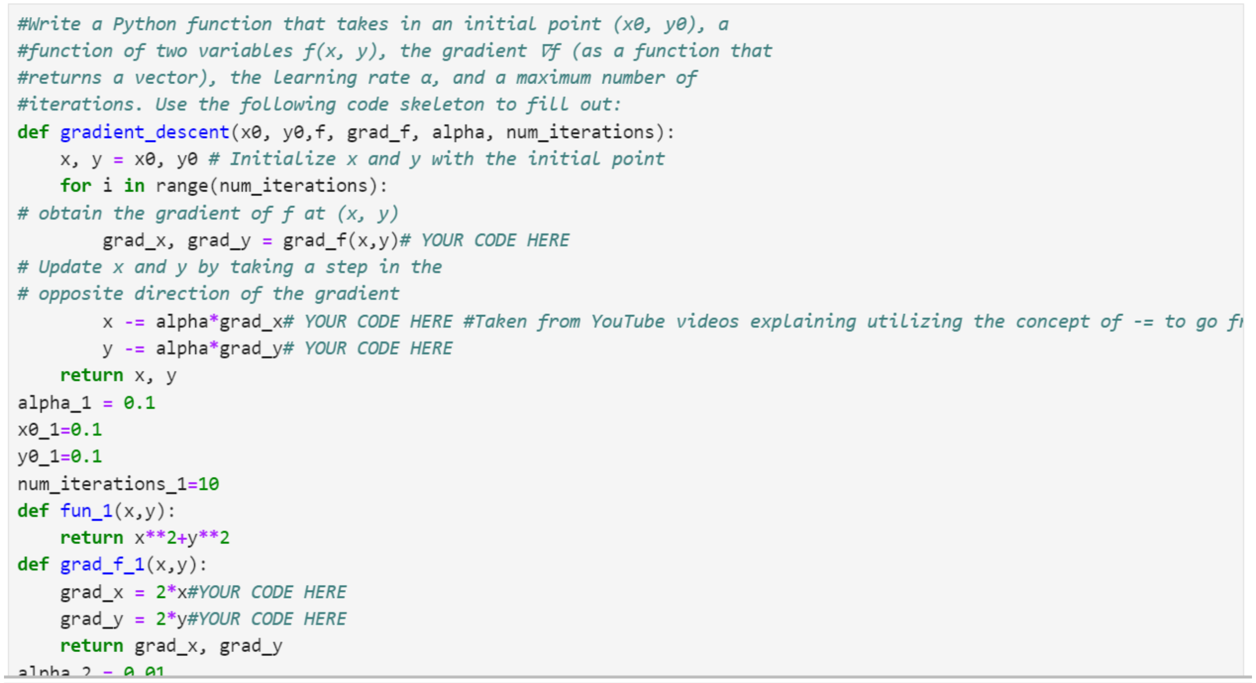
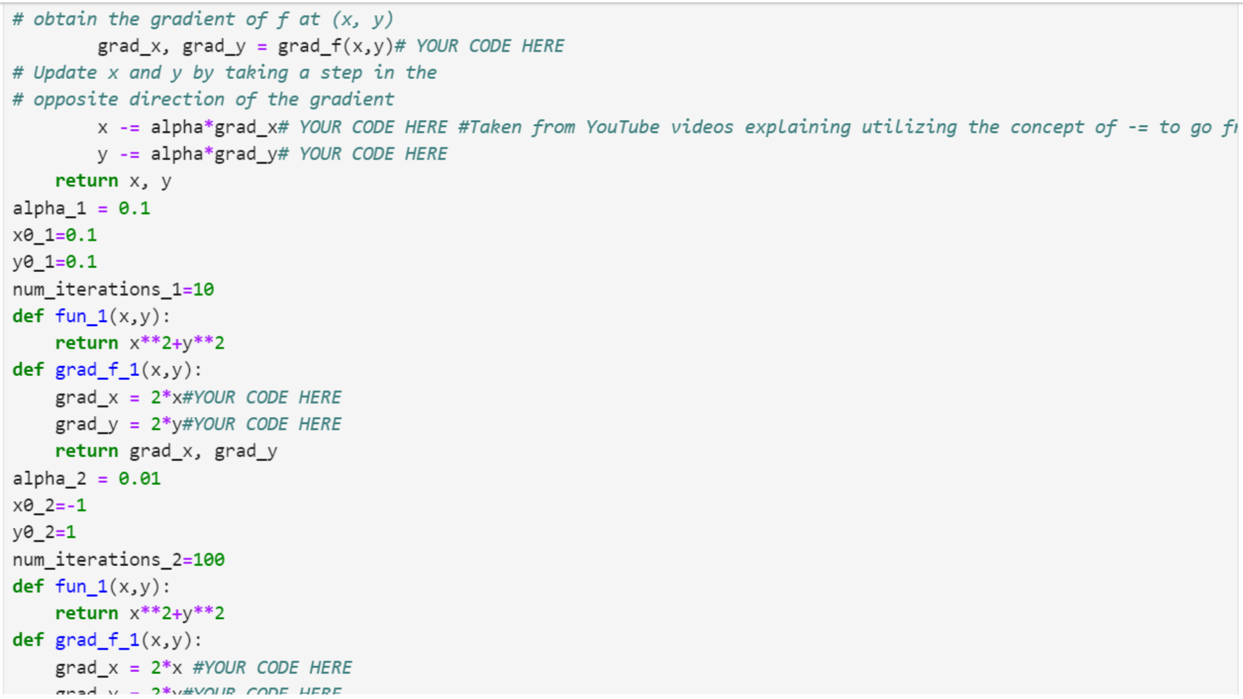
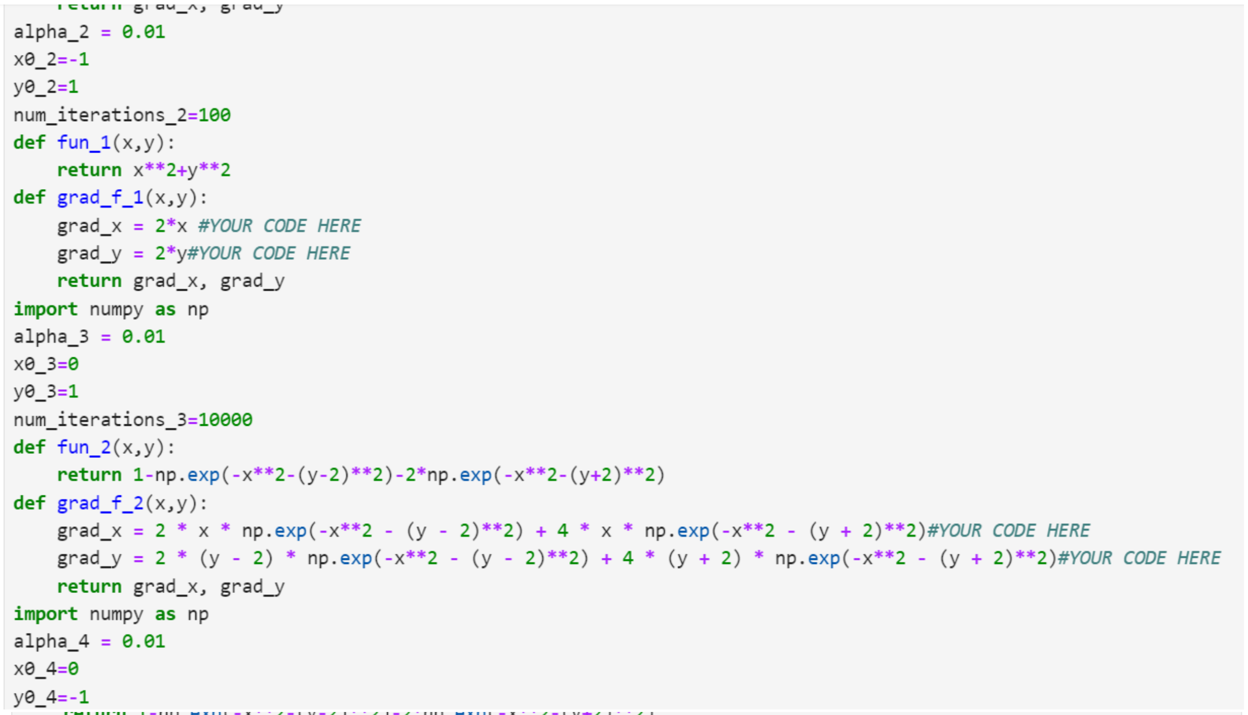


```python
#Write a Python function that takes in an initial point (x0, y0), a
#function of two variables f(x, y), the gradient ∇f (as a function that
#returns a vector), the learning rate α, and a maximum number of
#iterations. Use the following code skeleton to fill out:
def gradient_descent(x0, y0,f, grad_f, alpha, num_iterations):
    x, y = x0, y0 # Initialize x and y with the initial point
    for i in range(num_iterations):
# obtain the gradient of f at (x, y)
        grad_x, grad_y = grad_f(x,y)# YOUR CODE HERE
# Update x and y by taking a step in the
# opposite direction of the gradient
        x -= alpha*grad_x# YOUR CODE HERE #Taken from YouTube videos explaining utilizing the concept of -= to go f
        y -= alpha*grad_y# YOUR CODE HERE
    return x, y
alpha_1 = 0.1
x0_1=0.1
y0_1=0.1
num_iterations_1=10
def fun_1(x,y):
    return x**2+y**2
def grad_f_1(x,y):
    grad_x = 2*x#YOUR CODE HERE
    grad_y = 2*y#YOUR CODE HERE
    return grad_x, grad_y
alpha_2 = 0.01
```

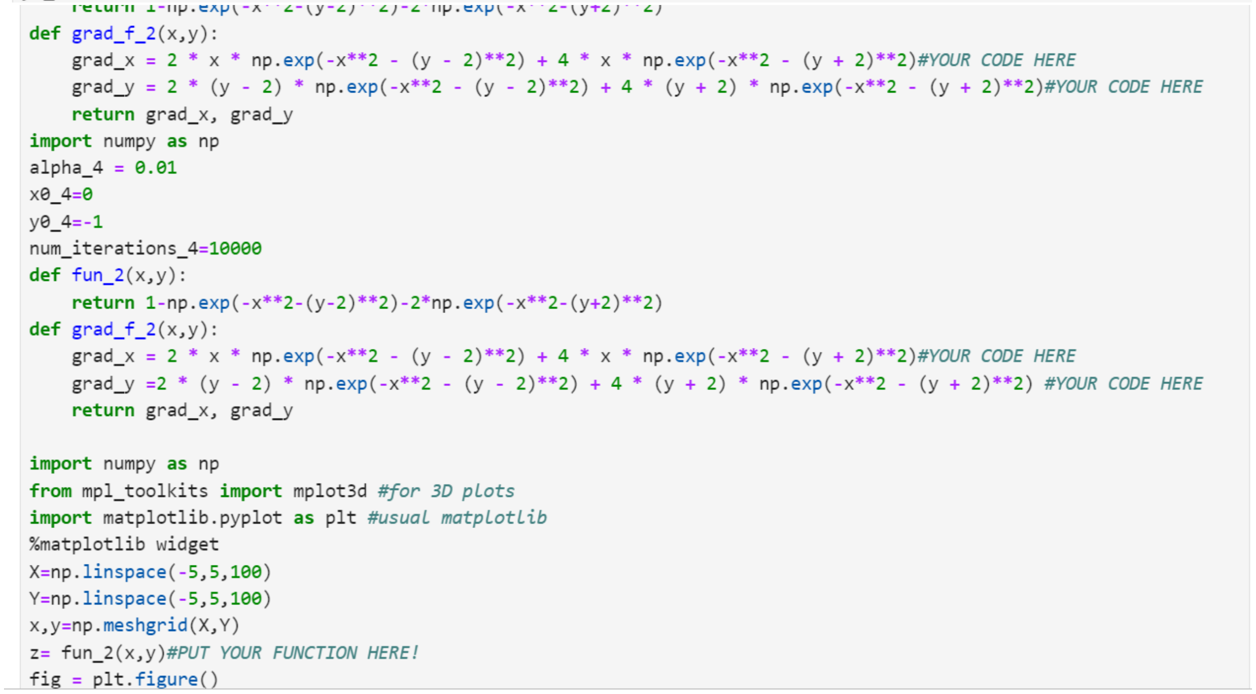


```python
# obtain the gradient of f at (x, y)
        grad_x, grad_y = grad_f(x,y)# YOUR CODE HERE
# Update x and y by taking a step in the
# opposite direction of the gradient
        x -= alpha*grad_x# YOUR CODE HERE #Taken from YouTube videos explaining utilizing the concept of -= to go f
        y -= alpha*grad_y# YOUR CODE HERE
    return x, y
alpha_1 = 0.1
x0_1=0.1
y0_1=0.1
num_iterations_1=10
def fun_1(x,y):
    return x**2+y**2
def grad_f_1(x,y):
    grad_x = 2*x#YOUR CODE HERE
    grad_y = 2*y#YOUR CODE HERE
    return grad_x, grad_y
alpha_2 = 0.01
```

```python
x0_2=-1
y0_2=1
num_iterations_2=100
def fun_1(x,y):
    return x**2+y**2
def grad_f_1(x,y):
    grad_x = 2*x #YOUR CODE HERE
    grad_y = 2*y#YOUR CODE HERE
```

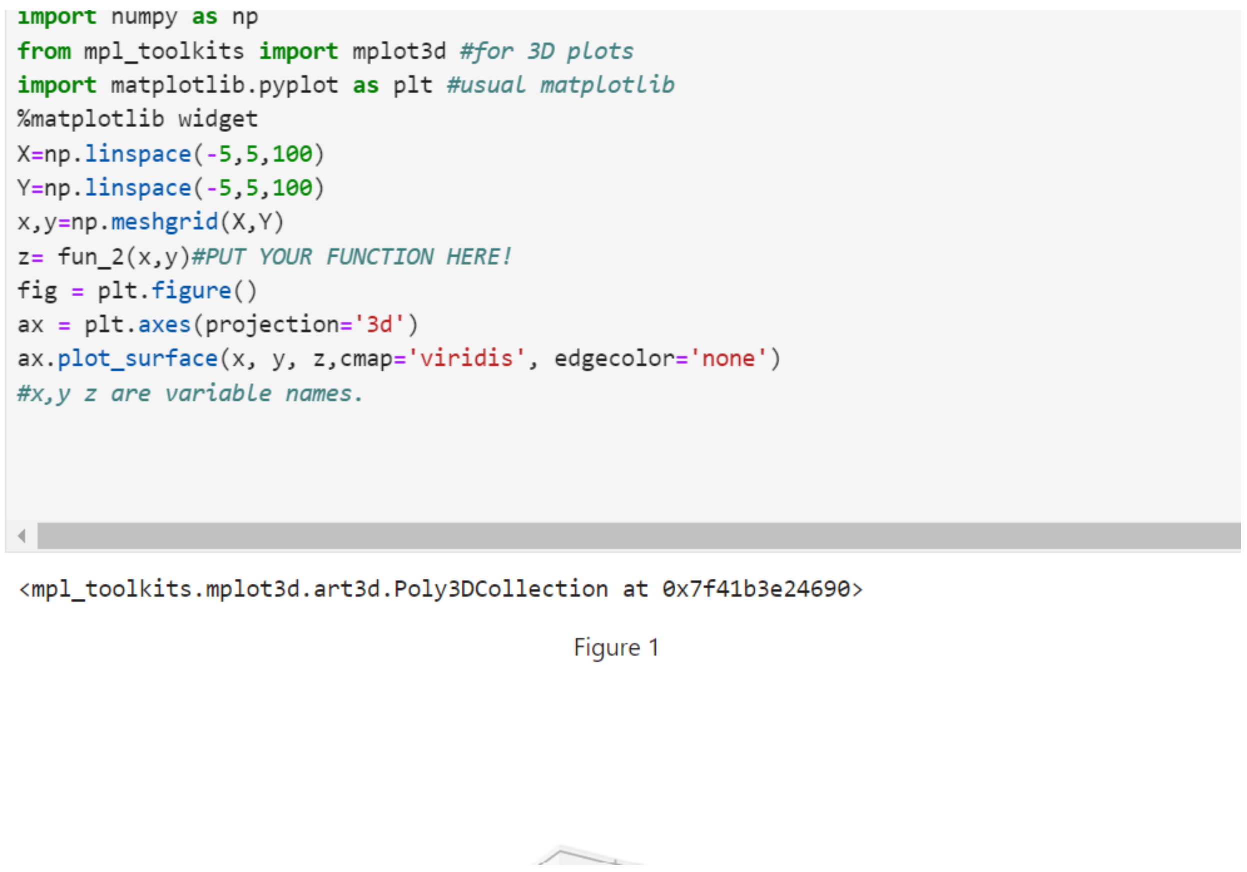


```python
    return grad_x, grad_y
alpha_2 = 0.01
x0_2=-1
y0_2=1
num_iterations_2=100
def fun_1(x,y):
    return x**2+y**2
def grad_f_1(x,y):
    grad_x = 2*x #YOUR CODE HERE
    grad_y = 2*y#YOUR CODE HERE
    return grad_x, grad_y
import numpy as np
alpha_3 = 0.01
x0_3=0
y0_3=1
num_iterations_3=10000
def fun_2(x,y):
    return 1-np.exp(-x**2-(y-2)**2)-2*np.exp(-x**2-(y+2)**2)
def grad_f_2(x,y):
    grad_x = 2 * x * np.exp(-x**2 - (y - 2)**2) + 4 * x * np.exp(-x**2 - (y + 2)**2)#YOUR CODE HERE
    grad_y = 2 * (y - 2) * np.exp(-x**2 - (y - 2)**2) + 4 * (y + 2) * np.exp(-x**2 - (y + 2)**2)#YOUR CODE HERE
    return grad_x, grad_y
import numpy as np
alpha_4 = 0.01
x0_4=0
y0_4=-1
```



```python
    return 1-np.exp(-x**2-(y-2)**2)-2*np.exp(-x**2-(y+2)**2)
def grad_f_2(x,y):
    grad_x = 2 * x * np.exp(-x**2 - (y - 2)**2) + 4 * x * np.exp(-x**2 - (y + 2)**2)#YOUR CODE HERE
    grad_y = 2 * (y - 2) * np.exp(-x**2 - (y - 2)**2) + 4 * (y + 2) * np.exp(-x**2 - (y + 2)**2)#YOUR CODE HERE
    return grad_x, grad_y
import numpy as np
alpha_4 = 0.01
x0_4=0
y0_4=-1
num_iterations_4=10000
def fun_2(x,y):
    return 1-np.exp(-x**2-(y-2)**2)-2*np.exp(-x**2-(y+2)**2)
def grad_f_2(x,y):
    grad_x = 2 * x * np.exp(-x**2 - (y - 2)**2) + 4 * x * np.exp(-x**2 - (y + 2)**2)#YOUR CODE HERE
    grad_y =2 * (y - 2) * np.exp(-x**2 - (y - 2)**2) + 4 * (y + 2) * np.exp(-x**2 - (y + 2)**2) #YOUR CODE HERE
    return grad_x, grad_y

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= fun_2(x,y)#PUT YOUR FUNCTION HERE!
fig = plt.figure()
```



```
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= fun_2(x,y)#PUT YOUR FUNCTION HERE!
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 0x7f41b3e24690>
```

Figure 1