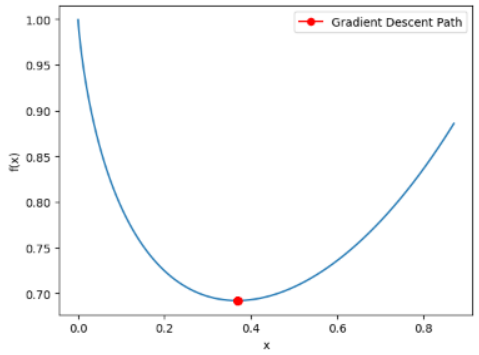
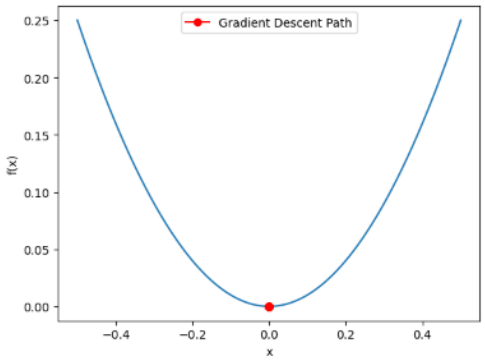
```python
import matplotlib.pyplot as plt #First import python languages to allow software to understand inputs
import numpy as np

def gradient_descent(f, df, learning_rate, initial_point): #Define gradient descent function to take in the function, derivative, learning rate(step size at each iteration, optimizing algorithm), and initial point(depends
    #on the function used)
    x_coords = [initial_point] #x and y coordinates are initialized
    y_coords = [f(initial_point)]

    for i in range(100): #Set my function to preform a loop for 100 iterations
        gradient = df(x_coords[-1])
        update = -learning_rate * gradient
        new_x = x_coords[-1] + update
        # Within these coding inputs, I allowed python to create the gradient descent of the x coordinate and update it for the next x coordinate.

        x_coords.append(new_x)
        y_coords.append(f(new_x))
        #Allowed these new x and y coordinates to be stored within the computer software to be accessed for the looping process.

    # Plot the function and the gradient descent path
    plot_range = np.linspace(min(x_coords) - 0.5, max(x_coords) + 0.5, 10000)
    function_range = f(plot_range)

    plt.plot(plot_range, function_range)
    plt.plot(x_coords, y_coords, marker='o', color='r', label='Gradient Descent Path')
    plt.xlabel('x')
    plt.ylabel('f(x)')
    plt.legend()
    plt.show()

    return round(x_coords[-1], 3), round(y_coords[-1], 3)
    #Asked python to output the final x and y coordinates following the gradient descent algorithm, to 3 decimal places.

# Define the function and its derivative, different derivative for each function
def f(x):
    return x**x

def df(x):
    return x**x * (np.log(x) + 1)

# Test the gradient descent function
initial_point = 0.37
learning_rate = 0.01 #Chose a small learning rate to showcase the steps towards the LOCAL max/min
final_point = gradient_descent(f, df, learning_rate, initial_point)
print(final_point)
```



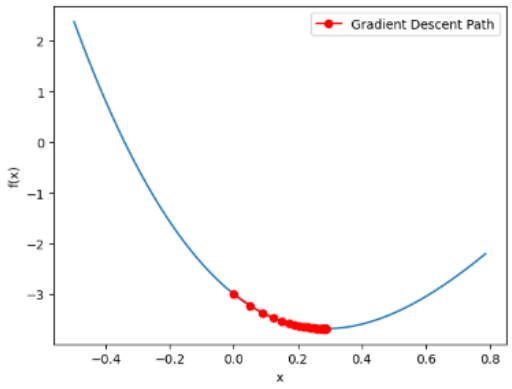

```
(0.368, 0.692)
```

```python
import matplotlib.pyplot as plt #First import python languages to allow software to understand inputs
import numpy as np

def gradient_descent(f, df, learning_rate, initial_point): #Define gradient descent function to take in the function, derivative, learning rate(step size at each iteration, optimizing algorithm), and initial point(depends
    #on the function used)
    x_coords = [initial_point] #x and y coordinates are initialized
    y_coords = [f(initial_point)]

    for i in range(100): #Set my function to preform a loop for 100 iterations
        gradient = df(x_coords[-1])
        update = -learning_rate * gradient
        new_x = x_coords[-1] + update
        # Within these coding inputs, I allowed python to create the gradient descent of the x coordinate and update it for the next x coordinate.

        x_coords.append(new_x)
        y_coords.append(f(new_x))
        #Allowed these new x and y coordinates to be stored within the computer software to be accessed for the looping process.

    # Plot the function and the gradient descent path
    plot_range = np.linspace(min(x_coords) - 0.5, max(x_coords) + 0.5, 10000)
    function_range = f(plot_range)

    plt.plot(plot_range, function_range)
    plt.plot(x_coords, y_coords, marker='o', color='r', label='Gradient Descent Path')
    plt.xlabel('x')
    plt.ylabel('f(x)')
    plt.legend()
    plt.show()

    return round(x_coords[-1], 3), round(y_coords[-1], 3)
    #Asked python to output the final x and y coordinates following the gradient descent algorithm, to 3 decimal places.

# Define the function and its derivative, different derivative for each function
def f(x):
    return x**2
def df(x):
    return 2*x

# Test the gradient descent function
initial_point = 0
learning_rate = 0.01 #Chose a small learning rate to showcase the steps towards the LOCAL max/min
final_point = gradient_descent(f, df, learning_rate, initial_point)
print(final_point)
```




```
(0.0, 0.0)
```

```python
import matplotlib.pyplot as plt #First import python languages to allow software to understand inputs
import numpy as np

def gradient_descent(f, df, learning_rate, initial_point): #Define gradient descent function to take in the function, derivative, learning rate(step size at each iteration, optimizing algorithm), and initial point(depends
    #on the function used)
    x_coords = [initial_point] #x and y coordinates are initialized
    y_coords = [f(initial_point)]

    for i in range(100): #Set my function to preform a loop for 100 iterations
        gradient = df(x_coords[-1])
        update = -learning_rate * gradient
        new_x = x_coords[-1] + update
        # Within these coding inputs, I allowed python to create the gradient descent of the x coordinate and update it for the next x coordinate.

        x_coords.append(new_x)
        y_coords.append(f(new_x))
        #Allowed these new x and y coordinates to be stored within the computer software to be accessed for the looping process.

    # Plot the function and the gradient descent path
    plot_range = np.linspace(min(x_coords) - 0.5, max(x_coords) + 0.5, 10000)
    function_range = f(plot_range)

    plt.plot(plot_range, function_range)
    plt.plot(x_coords, y_coords, marker='o', color='r', label='Gradient Descent Path')
    plt.xlabel('x')
    plt.ylabel('f(x)')
    plt.legend()
    plt.show()

    return round(x_coords[-1], 3), round(y_coords[-1], 3)
    #Asked python to output the final x and y coordinates following the gradient descent algorithm, to 3 decimal places.

# Define the function and its derivative, different derivative for each function
def f(x):
    return -3*x**3 + 10*x**2 - 5*x - 3
def df(x):
    return -9*x**2 + 20*x - 5

# Test the gradient descent function
initial_point = 0
learning_rate = 0.01 #Chose a small learning rate to showcase the steps towards the LOCAL max/min
final_point = gradient_descent(f, df, learning_rate, initial_point)
print(final_point)
```




```
(0.287, -3.682)
```

```python
import matplotlib.pyplot as plt #First import python languages to allow software to understand inputs
import numpy as np

def gradient_descent(f, df, learning_rate, initial_point): #Define gradient descent function to take in the function, derivative, learning rate(step size at each iteration, optimizing algorithm), and initial point(depends
    #on the function used)
    x_coords = [initial_point] #x and y coordinates are initialized
    y_coords = [f(initial_point)]

    for i in range(100): #Set my function to preform a loop for 100 iterations
        gradient = df(x_coords[-1])
        update = -learning_rate * gradient
        new_x = x_coords[-1] + update
        # Within these coding inputs, I allowed python to create the gradient descent of the x coordinate and update it for the next x coordinate.

        x_coords.append(new_x)
        y_coords.append(f(new_x))
        #Allowed these new x and y coordinates to be stored within the computer software to be accessed for the looping process.

    # Plot the function and the gradient descent path
    plot_range = np.linspace(min(x_coords) - 0.5, max(x_coords) + 0.5, 10000)
    function_range = f(plot_range)

    plt.plot(plot_range, function_range)
    plt.plot(x_coords, y_coords, marker='o', color='r', label='Gradient Descent Path')
    plt.xlabel('x')
    plt.ylabel('f(x)')
    plt.legend()
    plt.show()

    return round(x_coords[-1], 3), round(y_coords[-1], 3)
    #Asked python to output the final x and y coordinates following the gradient descent algorithm, to 3 decimal places.

# Define the function and its derivative, different derivative for each function
def f(x):
    return np.abs(x)
def df(x):
    return 0 if x == 0 else print("No max or min") #No abosolute max or min for this function


# Test the gradient descent function
initial_point = 0.1
learning_rate = 0.01 #Chose a small learning rate to showcase the steps towards the absolute max/min
final_point = gradient_descent(f, df, learning_rate, initial_point)
print(final_point)
```

```
No max or min
```