```python
# -*- coding: utf-8 -*-
"""Portfolio work 1.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/13VI5oulIHeWE2kltUsK3datx4Sfrq3nv
"""

#QUESTION 1
import numpy as np
import matplotlib.pyplot as plt

def euler_projectile(v0, theta, dt=0.01, g=9.81):
    """
    Simulates projectile motion using the Euler method.

    Parameters:
    v0 : float  -> Initial speed (m/s)
    theta : float  -> Launch angle (degrees)
    dt : float  -> Time step (s), default 0.01
    g : float  -> Acceleration due to gravity (m/s^2), default 9.81
    """

    # Convert degrees to radians
    theta = np.radians(theta)

    # Initial conditions
    x, y = 0, 0  # Position (m)
    vx, vy = v0 * np.cos(theta), v0 * np.sin(theta)  # Velocity components (m/s)

    # Lists to store trajectory points
    x_vals, y_vals = [x], [y]

    # Euler's loop
    while y >= 0:  # Ensures it stops when projectile hits the ground
        x += vx * dt  # Update x position
        y += vy * dt  # Update y position

        vy -= g * dt  # Update y velocity due to gravity

        x_vals.append(x)
        y_vals.append(y)

    return x_vals, y_vals

# Defined parameters
initial_speed = 50  # m/s
launch_angle = 45  # degrees
time_step = 0.01  # seconds

# Run simulation
x_traj, y_traj = euler_projectile(initial_speed, launch_angle, time_step)

# Plot results
plt.figure(figsize=(8, 5))
plt.plot(x_traj, y_traj, label=f'v0={initial_speed} m/s, θ={launch_angle}°')
plt.xlabel("Horizontal Distance (m)")
plt.ylabel("Vertical Distance (m)")
```

```
plt.title("Projectile Motion using Euler Method")
plt.legend()
plt.grid()
plt.show()

#QUESTION 2
import numpy as np
import matplotlib.pyplot as plt

def euler_projectile(v0, theta, dt=0.01, g=9.81):
    """
    Simulates projectile motion using the Euler method.

    Parameters:
    v0 : float  -> Initial speed (m/s)
    theta : float  -> Launch angle (degrees)
    dt : float  -> Time step (s), default 0.01
    g : float  -> Acceleration due to gravity (m/s^2), default 9.81
    """
    theta = np.radians(theta)  # Convert angle to radians

    # Initial conditions
    x, y = 0, 0
    vx, vy = v0 * np.cos(theta), v0 * np.sin(theta)

    x_vals, y_vals = [x], [y]

    while y >= 0:
        x += vx * dt  # Update x position
        y += vy * dt  # Update y position
        vy -= g * dt  # Update y velocity due to gravity
        x_vals.append(x)
        y_vals.append(y)

    return x_vals, y_vals, x_vals[-1]  # Return range as well

# Parameters
v0 = 10  # Initial speed in m/s
angles = [60, 30]  # Angles in degrees
dt = 0.01  # Time step

# Plot setup
plt.figure(figsize=(8, 5))
colors = ['b', 'r']

total_ranges = {}
for i, angle in enumerate(angles):
    x_traj, y_traj, x_range = euler_projectile(v0, angle, dt)
    plt.plot(x_traj, y_traj, colors[i], label=f'{angle}°')
    total_ranges[angle] = x_range
    print(f'Range for {angle}°: {x_range:.2f} meters')

# Plot formatting
plt.xlabel("Horizontal Distance (m)")
plt.ylabel("Vertical Distance (m)")
plt.title("Projectile Motion using Euler Method")
plt.legend()
plt.grid()
plt.show()
```

```
#QUESTION 3
import numpy as np
import matplotlib.pyplot as plt

def bouncing_ball_euler(y0, v0, dt, t_max, cor):
    g = 9.81  # Acceleration due to gravity (m/s^2)

    # Initial conditions
    y = y0
    vy = v0

    # Lists to store trajectory data
    y_values = [y]
    t_values = [0]

    # Time loop using Euler's method
    t = 0
    while t <= t_max:
        # Update position
        y = y + vy * dt

        # Update velocity
        vy = vy - g * dt

        # Check for bounce
        if y < 0:
            y = -y  # Reflect position
            vy = -vy * cor  # Reverse and dampen velocity

        # Store new positions
        y_values.append(y)
        t_values.append(t)

        # Increment time
        t += dt

    return t_values, y_values

# Parameters
y0 = 10            # Initial height (m)
v0 = 0             # Initial velocity (m/s)
dt = 0.01         # Time step (s)
t_max = 5          # Maximum simulation time (s)
cor = 0.8          # Coefficient of restitution (energy loss factor)

# Compute trajectory
t_vals, y_vals = bouncing_ball_euler(y0, v0, dt, t_max, cor)

# Plot results
plt.figure(figsize=(8, 5))
plt.plot(t_vals, y_vals, label="Bouncing Ball Path")
plt.xlabel("Time (s)")
plt.ylabel("Height (m)")
plt.title("Bouncing Ball using Euler's Method")
plt.legend()
plt.grid()
plt.show()
```