

```
[3] from google.colab import drive
drive.mount('/content/drive')

Mounted at /content/drive

# Define the quadratic function
def calculate_temperature(a, b, c, time):
    return a * (time ** 2) + b * time + c

# Hardcoded variables
a, b, c = 0.1, 2, 10
time = 5

# Calculate and display temperature
temperature = calculate_temperature(a, b, c, time)
print(f"Temperature at time {time} hours with hardcoded variables: {temperature}")

Temperature at time 5 hours with hardcoded variables: 22.5
```

```
# Accept coefficients from user input
a = float(input("Enter coefficient a: "))
b = float(input("Enter coefficient b: "))
c = float(input("Enter coefficient c: "))
time = float(input("Enter time: "))





# Calculate and display temperature
temperature = calculate_temperature(a, b, c, time)
print(f"Temperature at time {time} hours with keyboard input: {temperature}")

Enter coefficient a: 0.1
Enter coefficient b: 2
Enter coefficient c: 10
Enter time: 5
Temperature at time 5.0 hours with keyboard input: 22.5
```

My Drive > SE\_LAB > se

✓ ☰ ⓘ

Type ▾ People ▾ Modified ▾

Name	Owner	Last mo...	File size	
 multiple.txt	 me	10:09 PM	45 bytes	⋮
 single.txt	 me	10:09 PM	15 bytes	⋮

Files

SE\_LAB

SE\_LAB EXP 2.ipynb

SE\_LAB EXP 3.ipynb

multiple.txt

multiple\_inputs.txt

single.txt

single\_input.txt

archive (4)

+ Code + Text

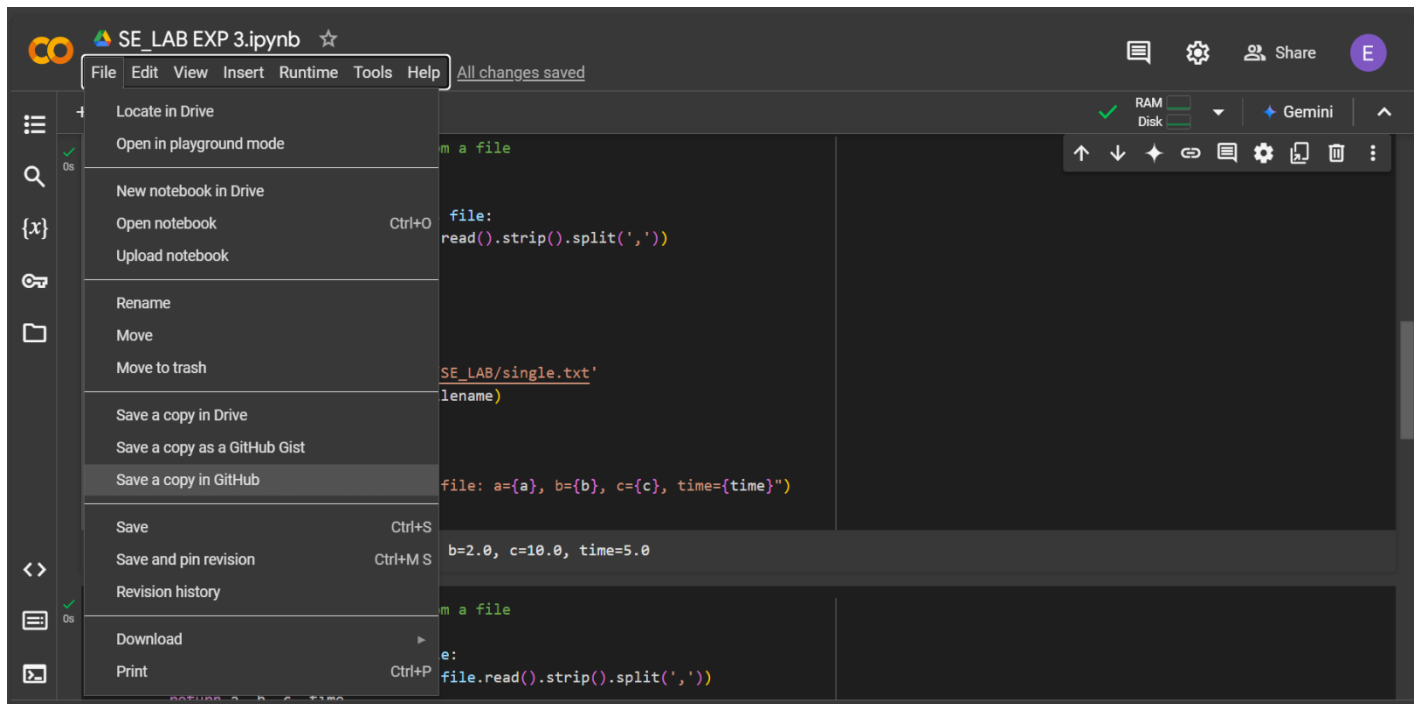
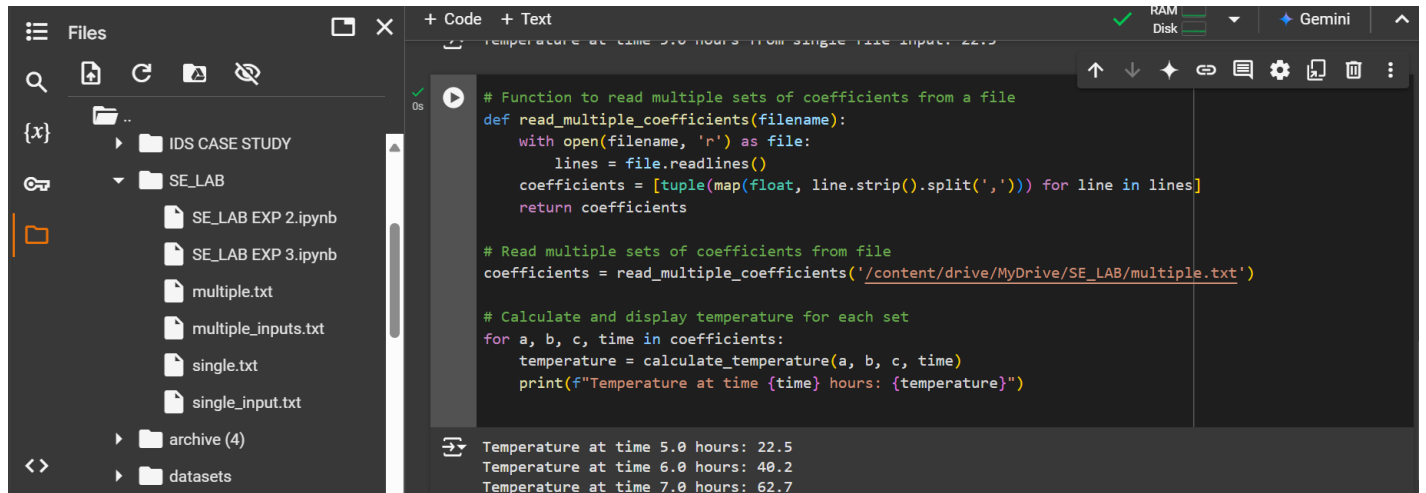
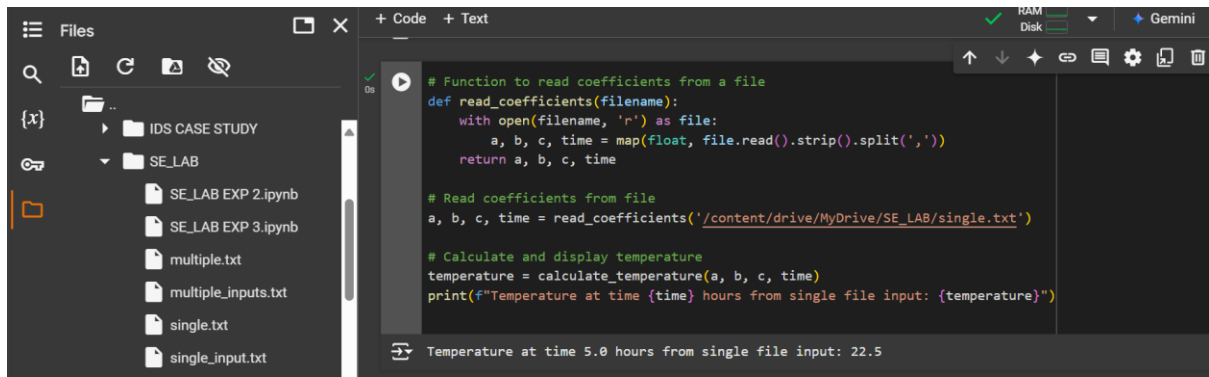
```
# Function to read coefficients from a file
def read_coefficients(filename):
    try:
        with open(filename, 'r') as file:
            return map(float, file.read().strip().split(','))
    except Exception as e:
        print(f"Error: {e}")
        return None

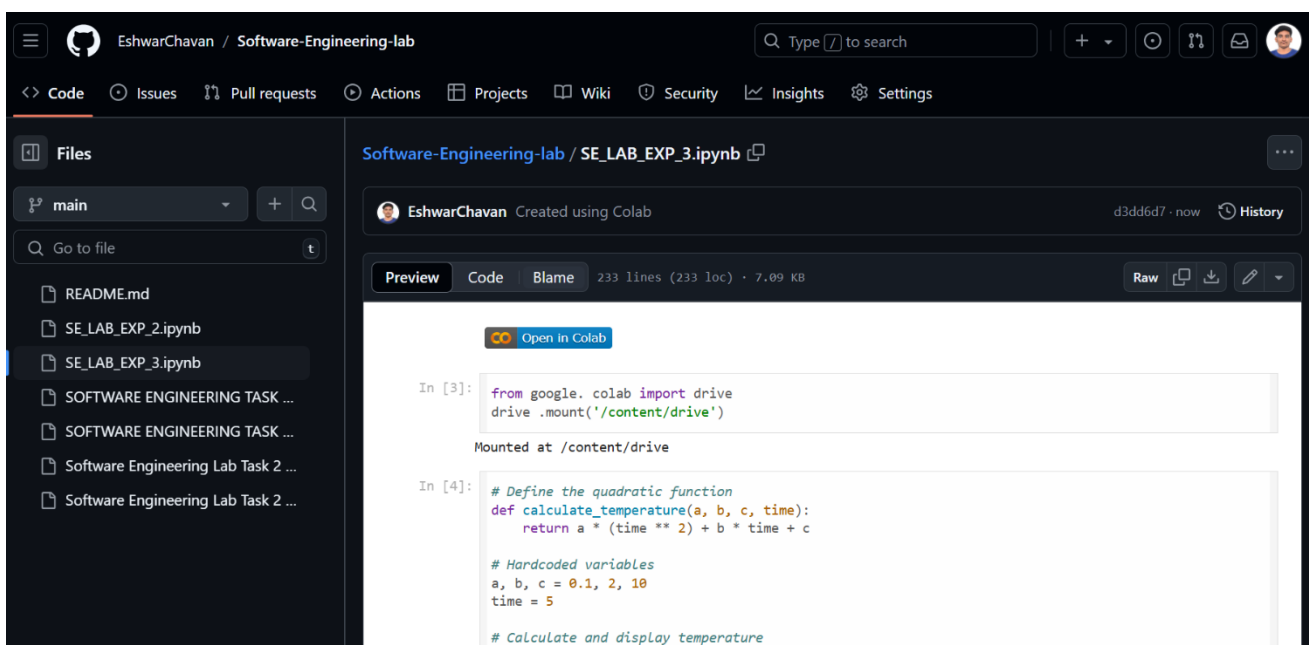
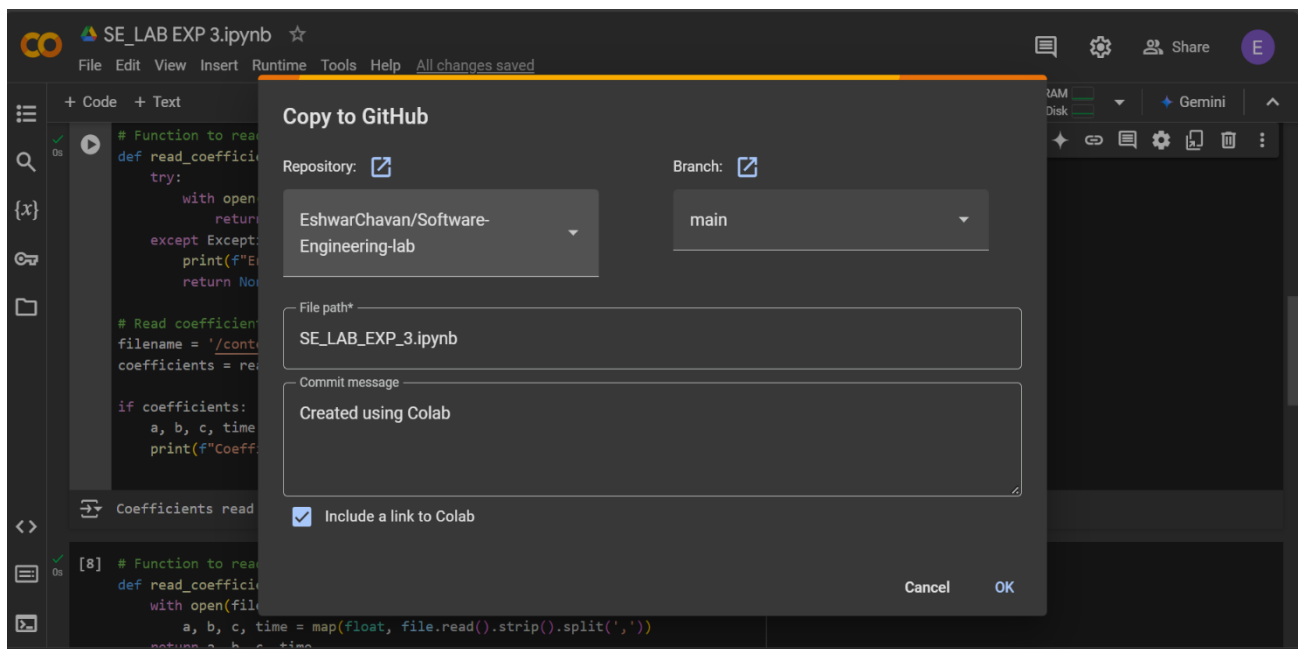
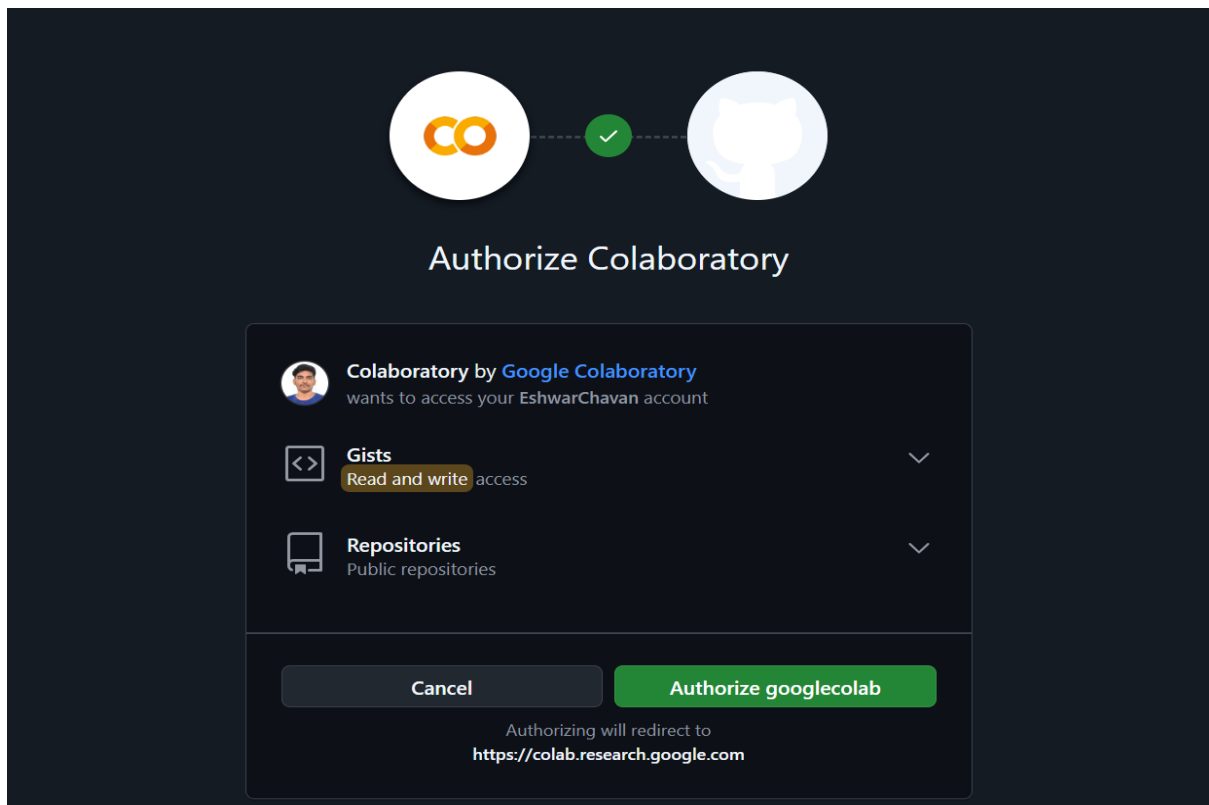
# Read coefficients from file
filename = '/content/drive/MyDrive/SE_LAB/single.txt'
coefficients = read_coefficients(filename)

if coefficients:
    a, b, c, time = coefficients
    print(f"Coefficients read from file: a={a}, b={b}, c={c}, time={time}")

Coefficients read from file: a=0.1, b=2.0, c=10.0, time=5.0
```

RAM Disk Gemini





## 1. Import Libraries

✓  
0s

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

✓  
0s

```
def quadratic(x, a=-1, b=0, c=10):
    return a * x**2 + b * x + c
```

▶

```
stages = ["Requirements", "Design", "Implementation", "Verification", "Maintenance"]

# Generate equally spaced x positions
x_positions = np.linspace(-2, 2, len(stages))

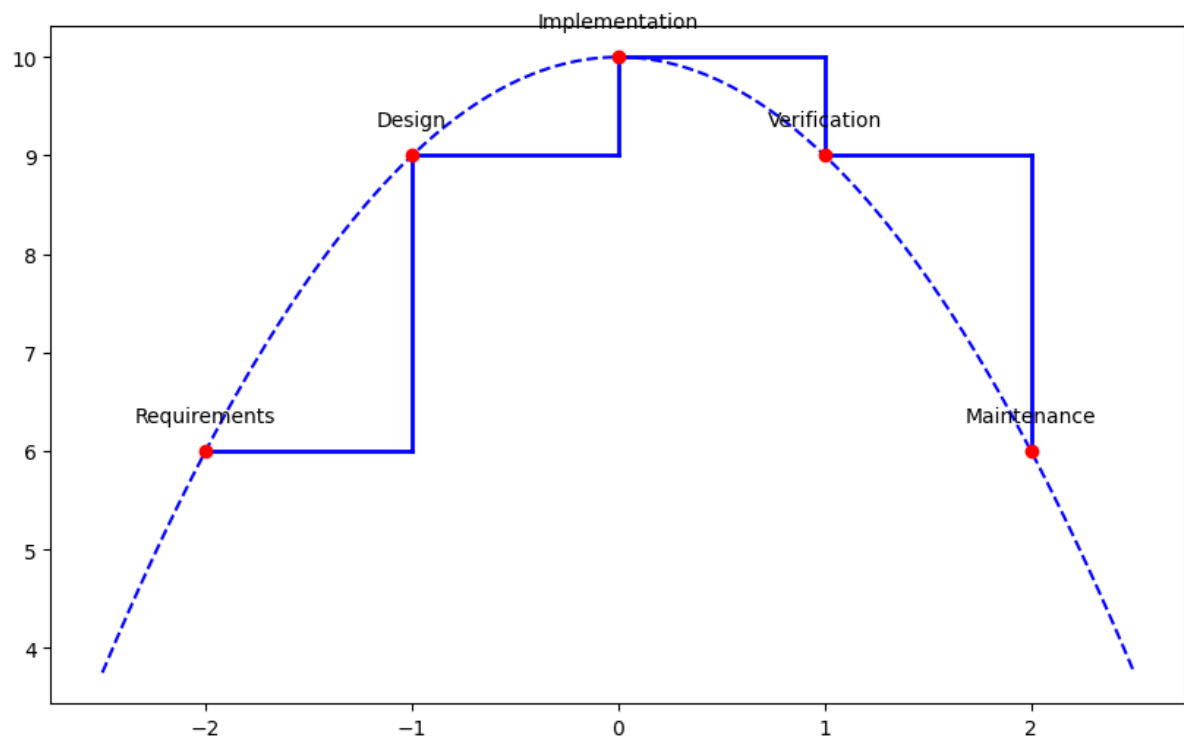
# Compute y positions using the quadratic function
y_positions = quadratic(x_positions) # Assuming quadratic function is defined elsewhere

# Plot the quadratic curve
x_curve = np.linspace(-2.5, 2.5, 500)
y_curve = quadratic(x_curve)

plt.figure(figsize=(10, 6))
plt.plot(x_curve, y_curve, color="blue", linestyle="--", label="Quadratic Curve")

# Add markers and connect them with vertical lines to create the waterfall effect
for i in range(len(stages)):
    if i > 0:
        # Draw a vertical drop to simulate a waterfall
        plt.plot([x_positions[i - 1], x_positions[i]], [y_positions[i - 1], y_positions[i]], color="blue", linewidth=2) # Horizontal segment
        plt.plot([x_positions[i], x_positions[i]], [y_positions[i - 1], y_positions[i]], color="blue", linewidth=2) # Vertical drop

    # Place markers and labels
    plt.scatter(x_positions[i], y_positions[i], color="red", zorder=5)
    plt.text(x_positions[i], y_positions[i] + 0.3, stages[i], ha="center", fontsize=10)
```



```

plt.plot(x_curve, y_curve, linestyle="--", color="green", label="Quadratic Curve")
plt.title("Model Representation Using Quadratic Curve", fontsize=12)
plt.xlabel("Stages of the Waterfall Model", fontsize=12)
plt.ylabel("Value", fontsize=12)
plt.axhline(0, color="black", linewidth=0.5, linestyle="--", alpha=0.7)
plt.axvline(0, color="black", linewidth=0.5, linestyle="--", alpha=0.7)
plt.grid(color="gray", linestyle="--", linewidth=0.5, alpha=0.7)
plt.legend()
plt.show()

```

