Lab – 30

Student Name: Navneet P

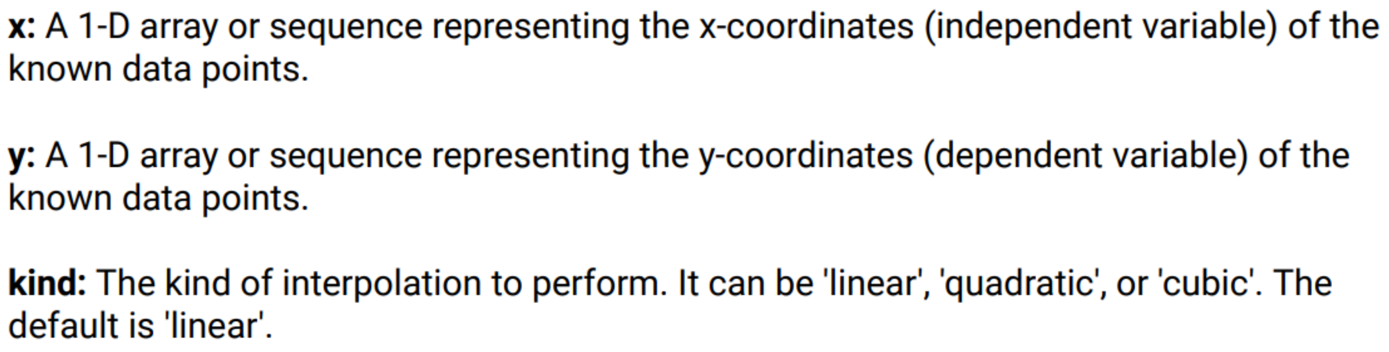
Student ID: AF0411619

Sci-Py Transform and Interpolation

Function and modules/submodules used:

1. **Scipy.interpolate**: this sub-module contains spline functions and classes, 1-D and multidimensional (univariate and multivariate) interpolation classes, Lagrange and Taylor polynomial interpolators, and wrappers for FITPACK and DFITPACK functions.

2. **Interp1d:** interp1d is a function in SciPy used for one-dimensional interpolation. It's a part of the scipy.interpolate module. This function creates an interpolation object that can be used to interpolate values for points that fall between the given data points.



x: A 1-D array or sequence representing the x-coordinates (independent variable) of the known data points.

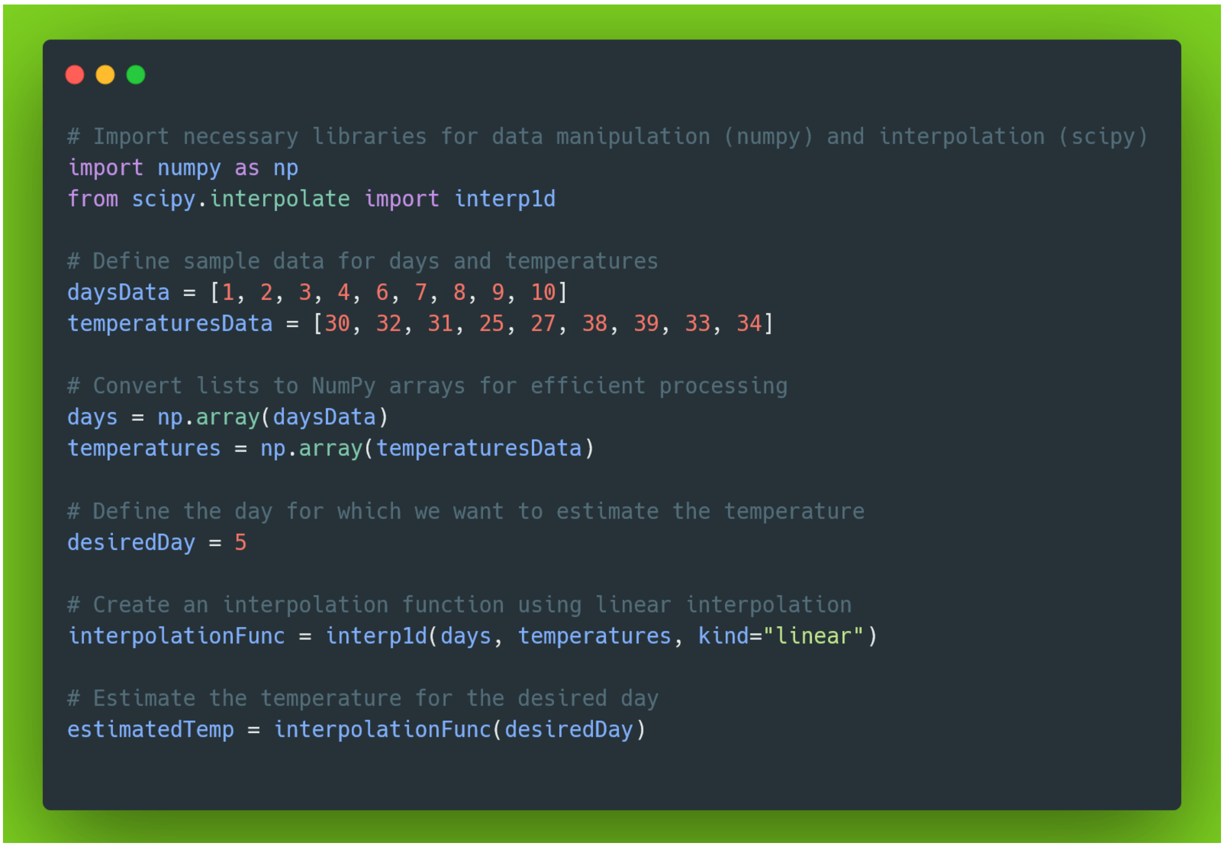
y: A 1-D array or sequence representing the y-coordinates (dependent variable) of the known data points.

kind: The kind of interpolation to perform. It can be 'linear', 'quadratic', or 'cubic'. The default is 'linear'.

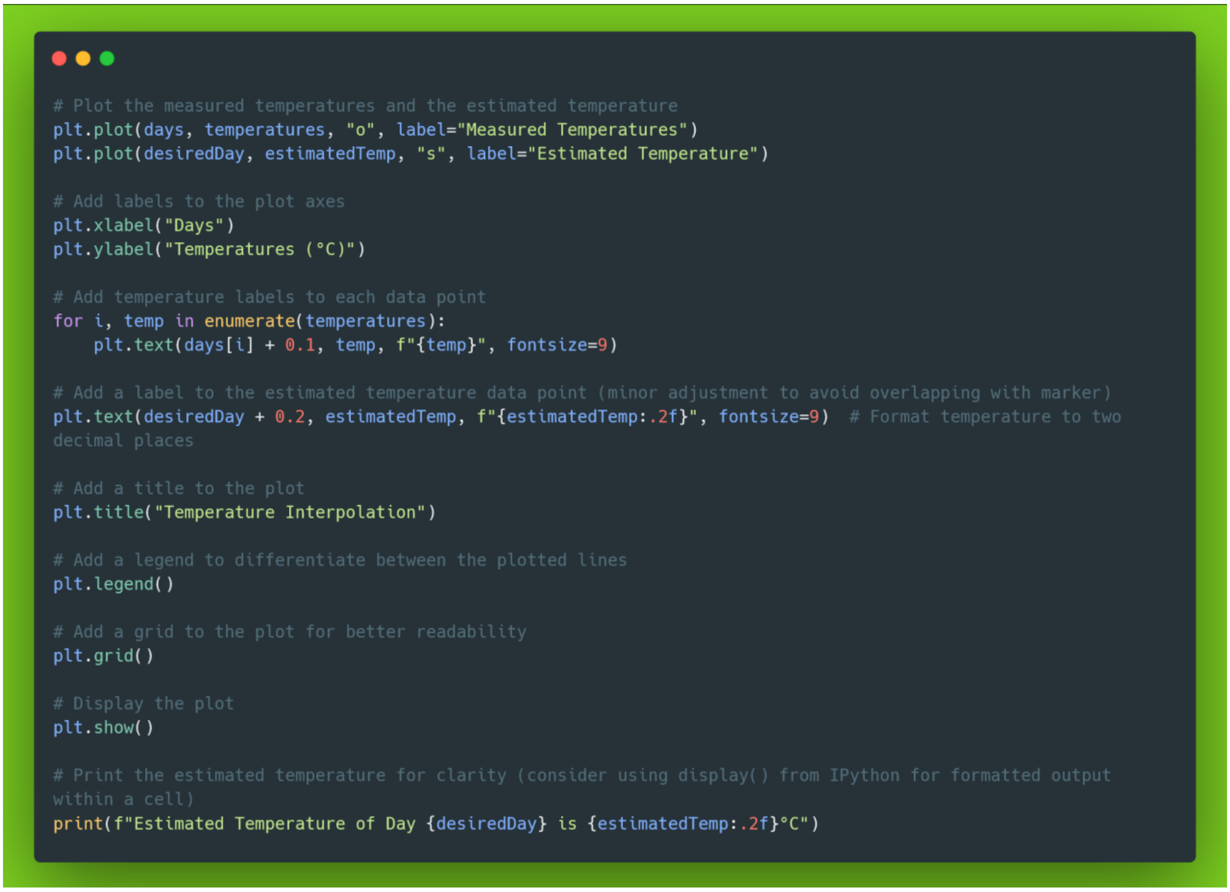
3. **Plt.text**: used to add data labels to the points on the graph

Q1. You are given the temperatures of 10 days but the 5th Day's temperature is not known, using the interpolation function in scipy, calculate the value of Day 5th Temperature.

Solution:

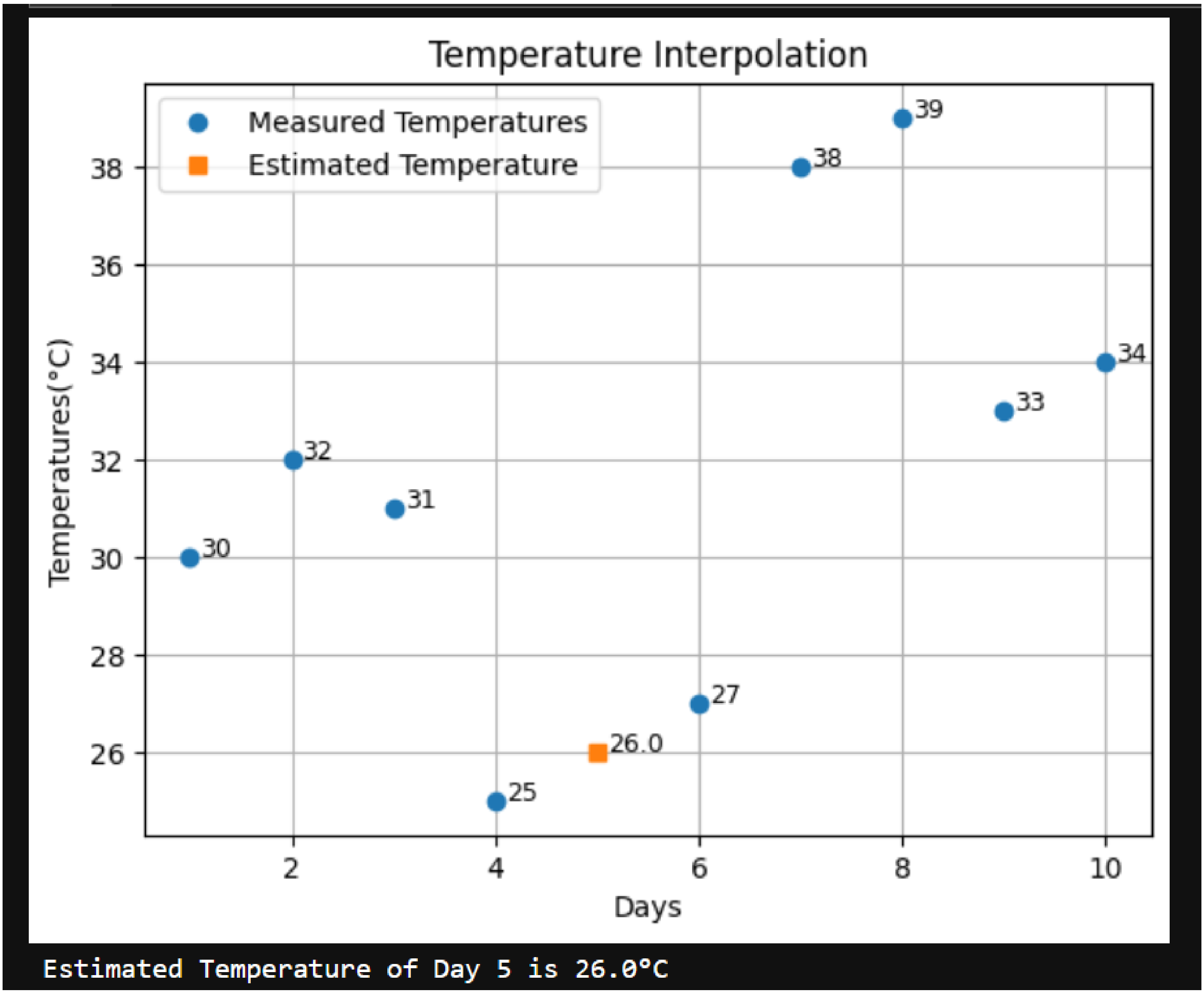


```python
# Import necessary libraries for data manipulation (numpy) and interpolation (scipy)
import numpy as np
from scipy.interpolate import interp1d

# Define sample data for days and temperatures
daysData = [1, 2, 3, 4, 6, 7, 8, 9, 10]
temperaturesData = [30, 32, 31, 25, 27, 38, 39, 33, 34]

# Convert lists to NumPy arrays for efficient processing
days = np.array(daysData)
temperatures = np.array(temperaturesData)

# Define the day for which we want to estimate the temperature
desiredDay = 5

# Create an interpolation function using linear interpolation
interpolationFunc = interp1d(days, temperatures, kind="linear")

# Estimate the temperature for the desired day
estimatedTemp = interpolationFunc(desiredDay)
```



```python
# Plot the measured temperatures and the estimated temperature
plt.plot(days, temperatures, "o", label="Measured Temperatures")
plt.plot(desiredDay, estimatedTemp, "s", label="Estimated Temperature")

# Add labels to the plot axes
plt.xlabel("Days")
plt.ylabel("Temperatures (°C)")

# Add temperature labels to each data point
for i, temp in enumerate(temperatures):
    plt.text(days[i] + 0.1, temp, f"{temp}", fontsize=9)

# Add a label to the estimated temperature data point (minor adjustment to avoid overlapping with marker)
plt.text(desiredDay + 0.2, estimatedTemp, f"{estimatedTemp:.2f}", fontsize=9)  # Format temperature to two
decimal places

# Add a title to the plot
plt.title("Temperature Interpolation")

# Add a legend to differentiate between the plotted lines
plt.legend()

# Add a grid to the plot for better readability
plt.grid()

# Display the plot
plt.show()

# Print the estimated temperature for clarity (consider using display() from IPython for formatted output
within a cell)
print(f"Estimated Temperature of Day {desiredDay} is {estimatedTemp:.2f}°C")
```

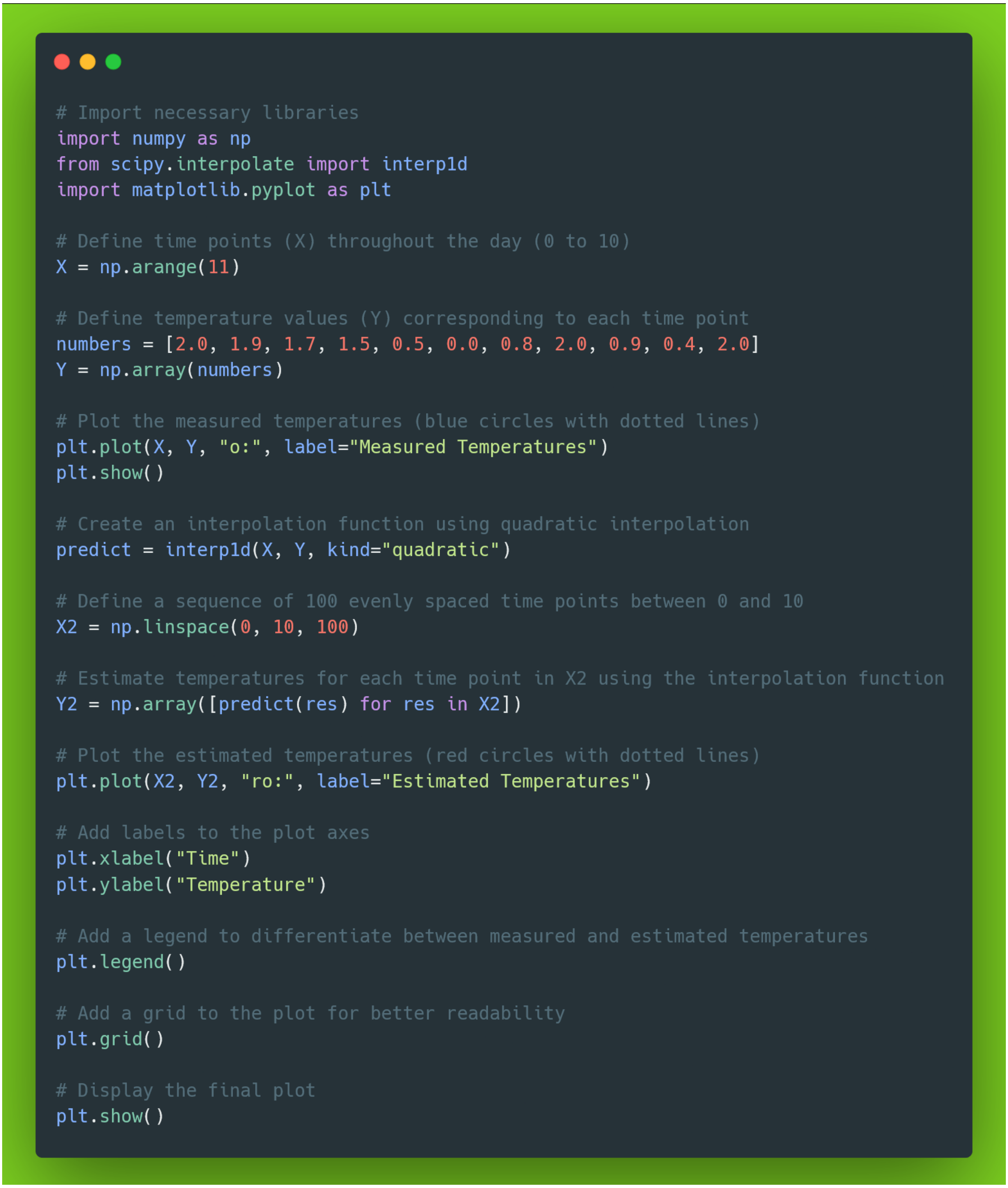
Output:



```
Estimated Temperature of Day 5 is 26.0°C
```

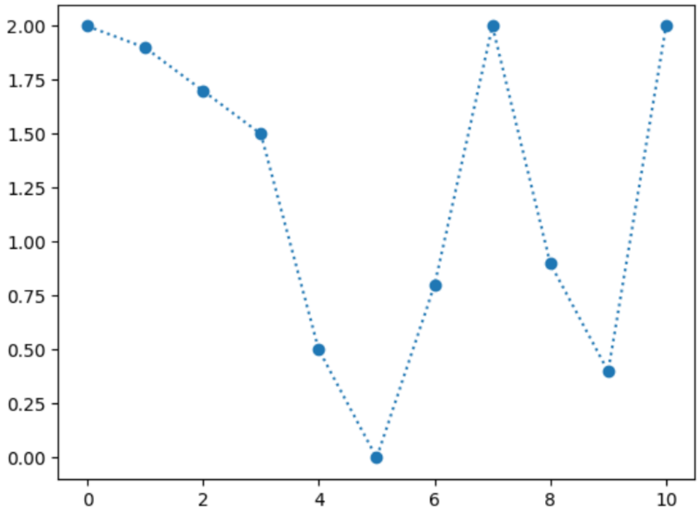
Q2. Interpolation question using a larger dataset.

Solution:

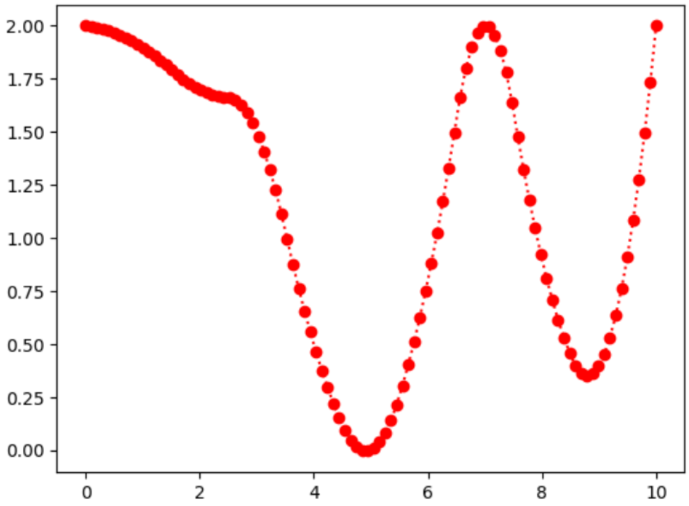


```python
# Import necessary libraries
import numpy as np
from scipy.interpolate import interp1d
import matplotlib.pyplot as plt

# Define time points (X) throughout the day (0 to 10)
X = np.arange(11)

# Define temperature values (Y) corresponding to each time point
numbers = [2.0, 1.9, 1.7, 1.5, 0.5, 0.0, 0.8, 2.0, 0.9, 0.4, 2.0]
Y = np.array(numbers)

# Plot the measured temperatures (blue circles with dotted lines)
plt.plot(X, Y, "o:", label="Measured Temperatures")
plt.show()

# Create an interpolation function using quadratic interpolation
predict = interp1d(X, Y, kind="quadratic")

# Define a sequence of 100 evenly spaced time points between 0 and 10
X2 = np.linspace(0, 10, 100)

# Estimate temperatures for each time point in X2 using the interpolation function
Y2 = np.array([predict(res) for res in X2])

# Plot the estimated temperatures (red circles with dotted lines)
plt.plot(X2, Y2, "ro:", label="Estimated Temperatures")

# Add labels to the plot axes
plt.xlabel("Time")
plt.ylabel("Temperature")

# Add a legend to differentiate between measured and estimated temperatures
plt.legend()

# Add a grid to the plot for better readability
plt.grid()

# Display the final plot
plt.show()
```

Output:



2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
2
4
6
8
10



2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
2
4
6
8
10