**Aim:** Simulate Discrete Time Sequences.

* **Write A Python Program to Generate the Given Signals and Plot them using Numpy and Matplotlib.**

1. Write a Python Programm To Plot Unit Impulse Signal.

**Programm:-**

import matplotlib.pyplot as plt

import numpy as np

def unit\_impulse(*length*, *position*):

    signal = np.zeros(length)

    signal[position] = 1

    return signal

*# Parameters*

start = -10  *# Start value of the x-axis range*

stop = 10  *# Stop value of the x-axis range*

step = 1  *# Step size*

*# Generate x-axis values*

x = np.arange(start, stop+step, step)

*# Generate unit impulse signal*

impulse\_signal = unit\_impulse(len(x), abs(start)//step)

*# Plot the signal*

plt.stem(x, impulse\_signal)

plt.xlabel('Time')

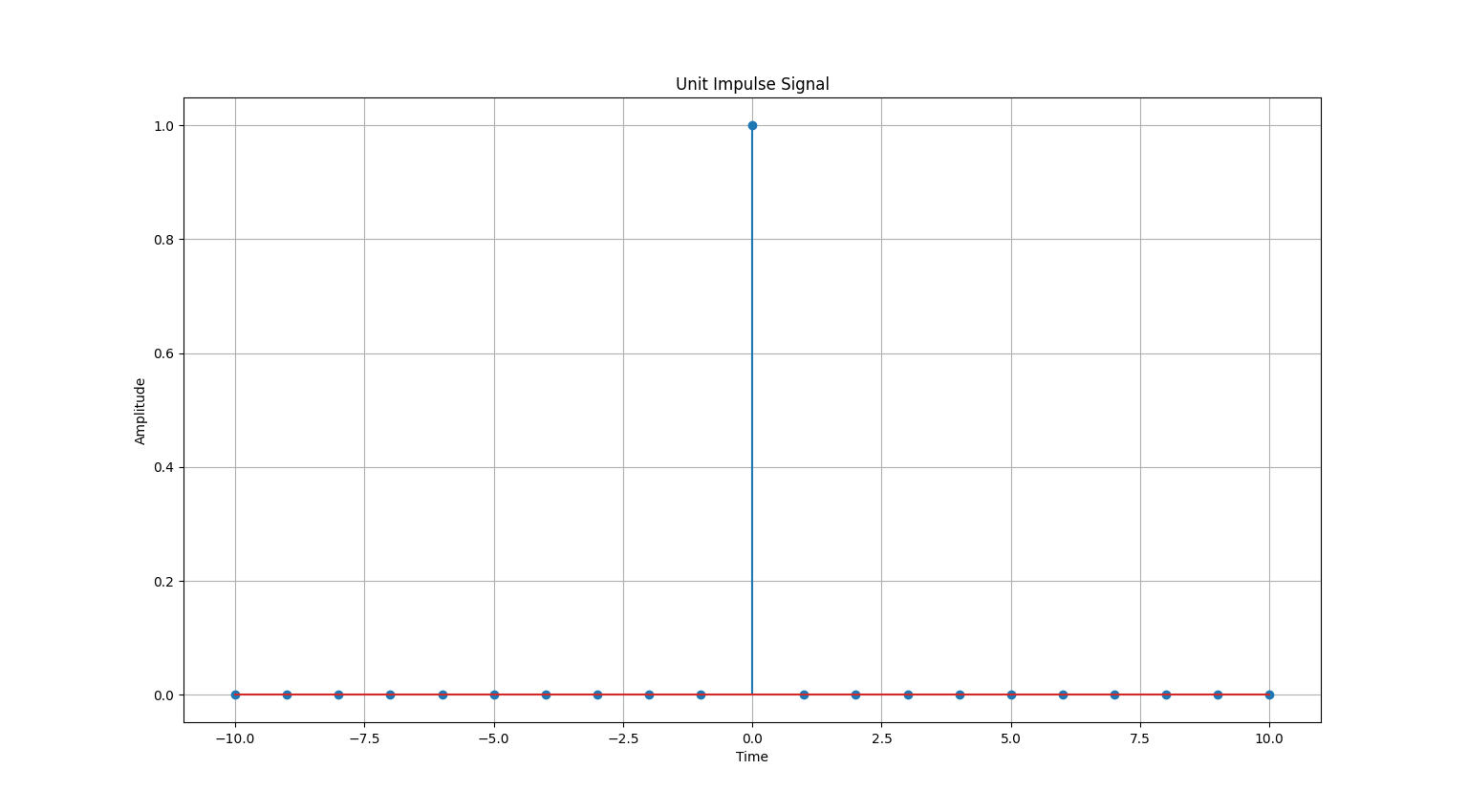
plt.ylabel('Amplitude')

plt.title('Unit Impulse Signal')

plt.grid(True)

plt.show()

**Output:-**



1. Write a Python Programm To Plot Unit Impulse Train.

**Programm:-**

import matplotlib.pyplot as plt

import numpy as np

def simulate\_impulse\_train(*signal\_length*, *period*):

    impulse\_train = np.zeros(signal\_length)

    for n in range(signal\_length):

        if n % period == 0:

            impulse\_train[n] = 1

    return impulse\_train

*# Define the parameters for the impulse train*

signal\_length = 100  *# Length of the impulse train*

period = 10  *# Period of the impulse train*

*# Simulate the impulse train*

impulse\_train = simulate\_impulse\_train(signal\_length, period)

*# Plot and display the impulse train*

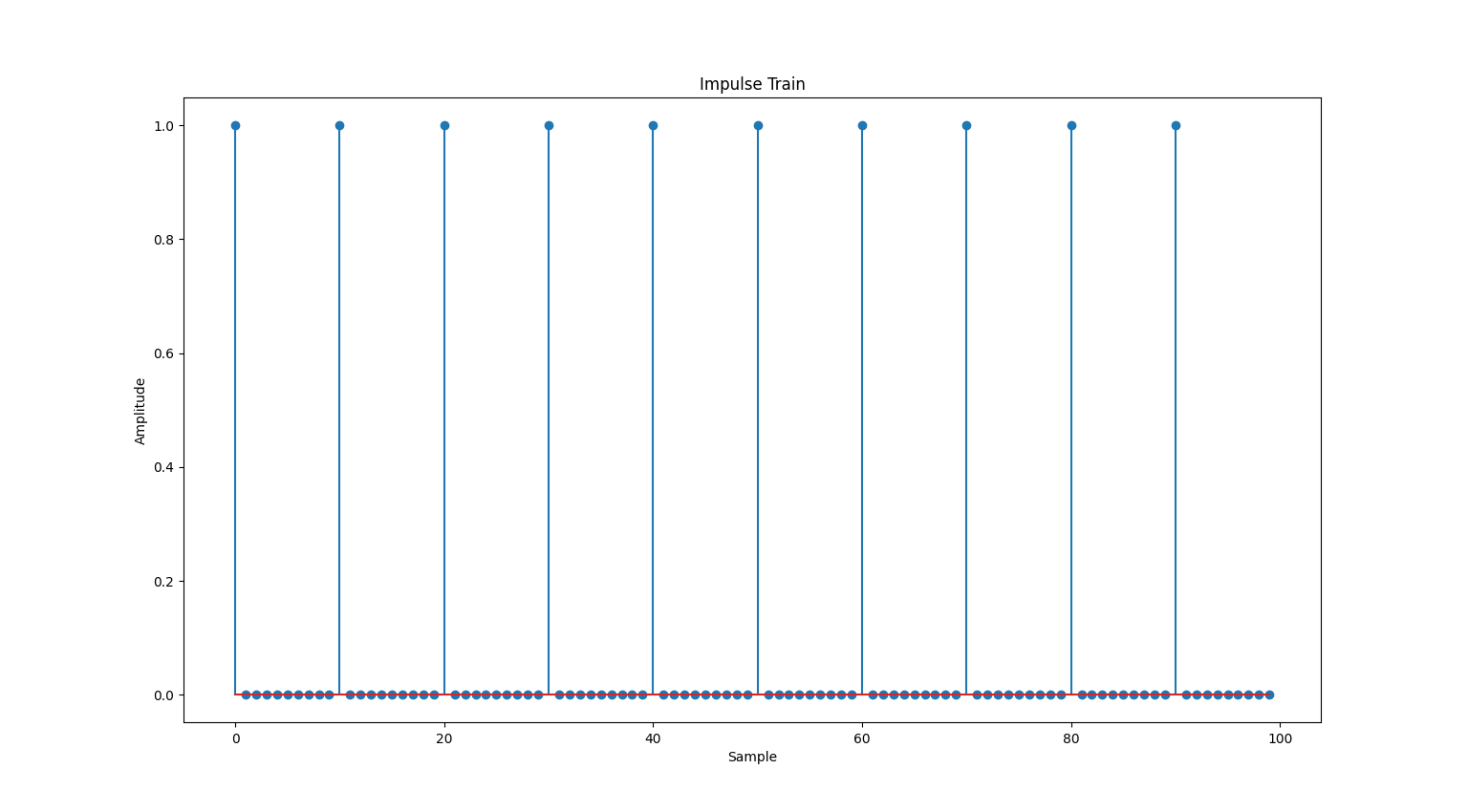
plt.stem(impulse\_train)

plt.title('Impulse Train')

plt.xlabel('Sample')

plt.ylabel('Amplitude')

plt.show()

**Output:-**

1. Write a Python Program to Simulate Continuous and Discrete Unit Step Signals.

**Programm:-**

import matplotlib.pyplot as plt

import numpy as np

def simulate\_continuous\_unit\_step(*time*):

    unit\_step = np.zeros\_like(time)

    unit\_step[time >= 0] = 1

    return unit\_step

def simulate\_discrete\_unit\_step(*num\_samples*):

    unit\_step = np.zeros(num\_samples)

    unit\_step[num\_samples // 2:] = 1

    return unit\_step

*# Define the time range for the continuous unit step signal*

time = np.linspace(-5, 5, 1000)  *# Time range from -5 to 5*

*# Simulate the continuous unit step signal*

continuous\_unit\_step = simulate\_continuous\_unit\_step(time)

*# Define the number of samples for the discrete unit step signal*

num\_samples = 20  *# Number of samples*

*# Simulate the discrete unit step signal*

discrete\_unit\_step = simulate\_discrete\_unit\_step(num\_samples)

*# Plot and display the continuous and discrete unit step signals*

plt.figure(*figsize*=(10, 6))

plt.subplot(2, 1, 1)

plt.plot(time, continuous\_unit\_step)

plt.title('Continuous Unit Step Signal')

plt.xlabel('Time')

plt.ylabel('Amplitude')

plt.subplot(2, 1, 2)

plt.stem(discrete\_unit\_step)

plt.title('Discrete Unit Step Signal')

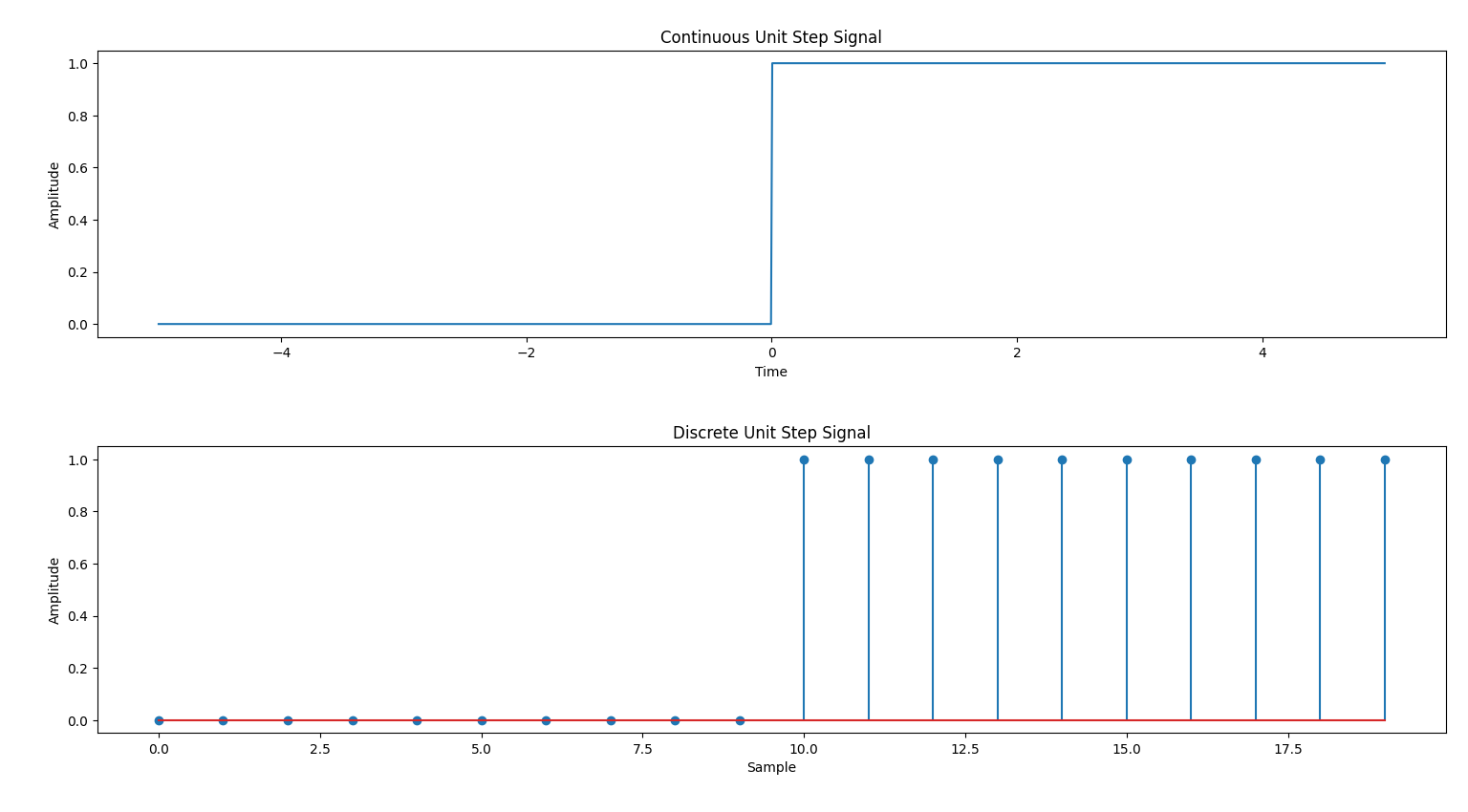
plt.xlabel('Sample')

plt.ylabel('Amplitude')

plt.tight\_layout()

plt.show()

**Output:-**



1. Write a Python Program to Simulate Continuous and Discrete Unit Ramp Signals.

**Programm:-**

import matplotlib.pyplot as plt

import numpy as np

def simulate\_continuous\_ramp(*time*, *slope*):

    ramp = np.zeros\_like(time)

    ramp[time >= 0] = slope \* time[time >= 0]

    return ramp

def simulate\_discrete\_ramp(*num\_samples*, *slope*):

    ramp = np.zeros(num\_samples)

    ramp[num\_samples // 2:] = slope \* np.arange(num\_samples // 2, num\_samples)

    return ramp

*# Define the time range for the continuous ramp signal*

time = np.linspace(-5, 5, 1000)  *# Time range from -5 to 5*

*# Define the number of samples and slope for the discrete ramp signal*

num\_samples = 20  *# Number of samples*

slope = 2  *# Slope of the ramp*

*# Simulate the continuous ramp signal*

continuous\_ramp = simulate\_continuous\_ramp(time, slope)

*# Simulate the discrete ramp signal*

discrete\_ramp = simulate\_discrete\_ramp(num\_samples, slope)

*# Plot and display the continuous and discrete ramp signals*

plt.figure(*figsize*=(10, 6))

plt.subplot(2, 1, 1)

plt.plot(time, continuous\_ramp)

plt.title('Continuous Ramp Signal')

plt.xlabel('Time')

plt.ylabel('Amplitude')

plt.subplot(2, 1, 2)

plt.stem(discrete\_ramp)

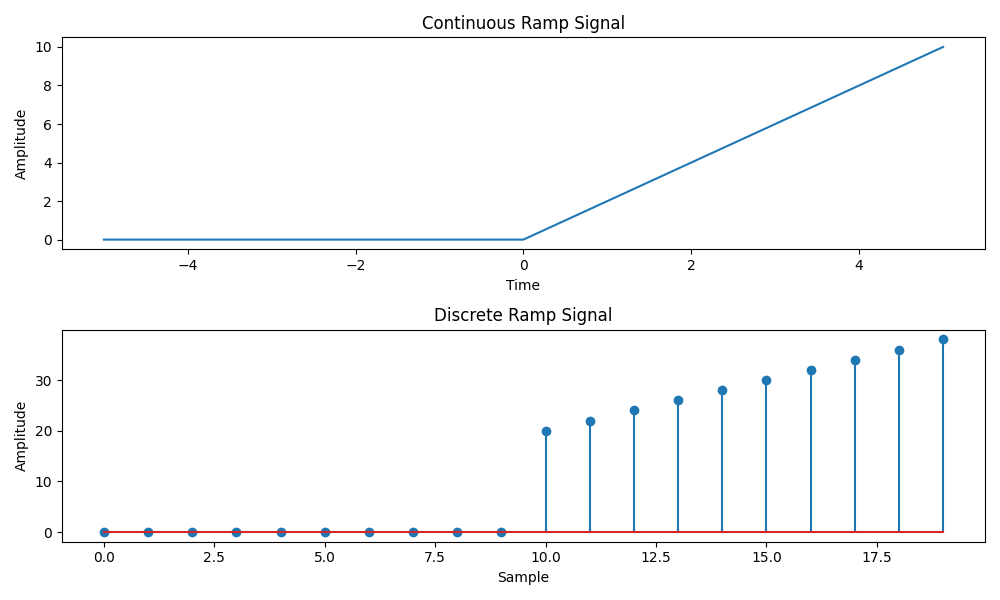
plt.title('Discrete Ramp Signal')

plt.xlabel('Sample')

plt.ylabel('Amplitude')

plt.tight\_layout()

plt.show()

**Output:-**

1. Write a Python Program to Simulate Continuous and Discrete Exponential Signals.

**Programm:-**

import matplotlib.pyplot as plt

import numpy as np

def simulate\_continuous\_exponential(*time*, *amplitude*, *coefficient*):

    exponential\_signal = amplitude \* np.exp(coefficient \* time)

    return exponential\_signal

def simulate\_discrete\_exponential(*num\_samples*, *amplitude*, *coefficient*):

    exponential\_signal = amplitude \* \

        np.exp(coefficient \* np.arange(num\_samples))

    return exponential\_signal

*# Define the time range for the continuous exponential signal*

time = np.linspace(0, 5, 1000)  *# Time range from 0 to 5*

*# Define the number of samples, initial amplitude, and coefficient for the*

*# discrete exponential signal*

num\_samples = 20  *# Number of samples*

amplitude = 2  *# Initial amplitude*

coefficient = -0.5  *# Exponential coefficient*

*# Simulate the continuous exponential signal*

continuous\_exponential = simulate\_continuous\_exponential(

    time, amplitude, coefficient)

*# Simulate the discrete exponential signal*

discrete\_exponential = simulate\_discrete\_exponential(

    num\_samples, amplitude, coefficient)

*# Plot and display the continuous and discrete exponential signals*

plt.figure(*figsize*=(10, 6))

plt.subplot(2, 1, 1)

plt.plot(time, continuous\_exponential)

plt.title('Continuous Exponential Signal')

plt.xlabel('Time')

plt.ylabel('Amplitude')

plt.subplot(2, 1, 2)

plt.stem(discrete\_exponential)

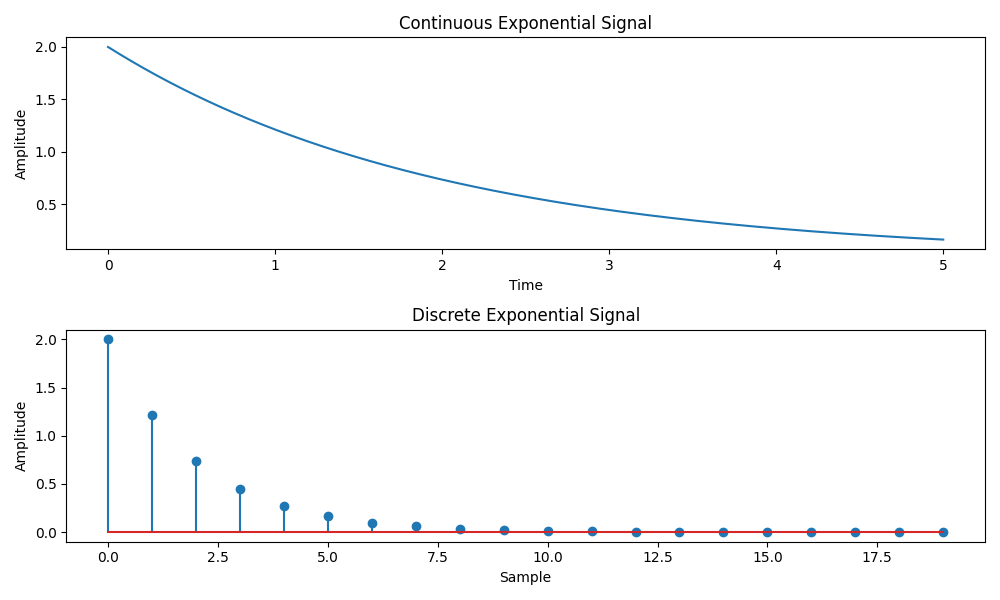
plt.title('Discrete Exponential Signal')

plt.xlabel('Sample')

plt.ylabel('Amplitude')

plt.tight\_layout()

plt.show()

**Output:-**

1. Write a Python Program to Simulate Continuous and Discrete Parabolic Signals.

**Programm:-**

import matplotlib.pyplot as plt

import numpy as np

def simulate\_continuous\_parabolic(*time*, *coefficients*):

    parabolic\_signal = np.polyval(coefficients, time)

    return parabolic\_signal

def simulate\_discrete\_parabolic(*num\_samples*, *coefficients*):

    parabolic\_signal = np.polyval(coefficients, np.arange(num\_samples))

    return parabolic\_signal

*# Define the time range for the continuous parabolic signal*

time = np.linspace(-5, 5, 1000)  *# Time range from -5 to 5*

*# Define the number of samples and coefficients for the discrete parabolic*

*# signal*

num\_samples = 20  *# Number of samples*

coefficients = [1, 2, 1]  *# Coefficients of the parabolic signal*

*# Simulate the continuous parabolic signal*

continuous\_parabolic = simulate\_continuous\_parabolic(time, coefficients)

*# Simulate the discrete parabolic signal*

discrete\_parabolic = simulate\_discrete\_parabolic(num\_samples, coefficients)

*# Plot and display the continuous and discrete parabolic signals*

plt.figure(*figsize*=(10, 6))

plt.subplot(2, 1, 1)

plt.plot(time, continuous\_parabolic)

plt.title('Continuous Parabolic Signal')

plt.xlabel('Time')

plt.ylabel('Amplitude')

plt.subplot(2, 1, 2)

plt.stem(discrete\_parabolic)

plt.title('Discrete Parabolic Signal')

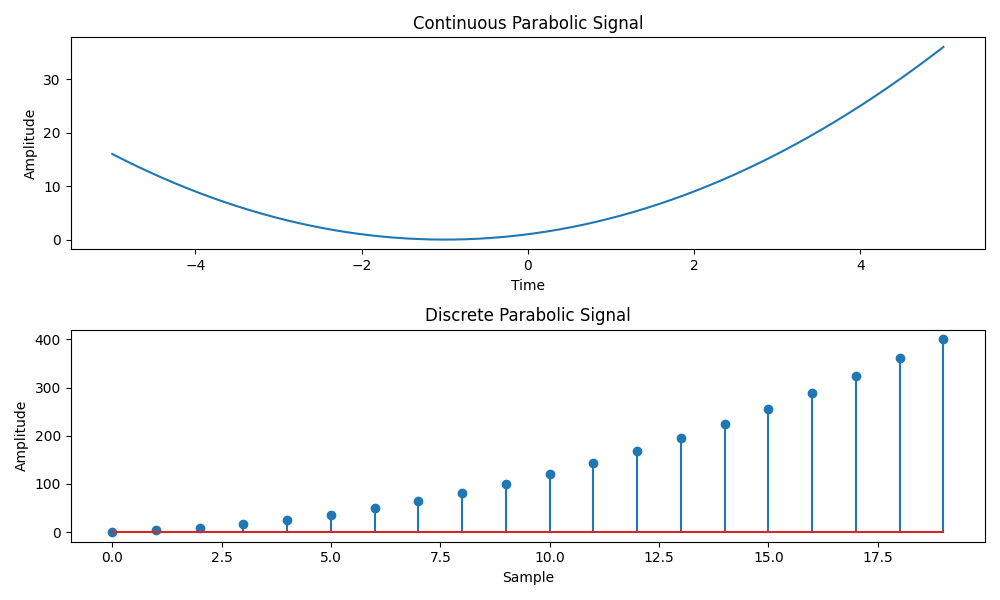
plt.xlabel('Sample')

plt.ylabel('Amplitude')

plt.tight\_layout()

plt.show()

**Output:-**



1. Write a Python Program to Simulate Continuous and Discrete Sine Wave Signals.

**Programm:-**

import matplotlib.pyplot as plt

import numpy as np

def simulate\_continuous\_sine\_wave(*time*, *amplitude*, *frequency*, *phase*):

    sine\_wave = amplitude \* np.sin(2 \* np.pi \* frequency \* time + phase)

    return sine\_wave

def simulate\_discrete\_sine\_wave(*num\_samples*, *sampling\_frequency*, *amplitude*,

*frequency*, *phase*):

    time = np.arange(num\_samples) / sampling\_frequency

    sine\_wave = amplitude \* np.sin(2 \* np.pi \* frequency \* time + phase)

    return sine\_wave

*# Define the time range for the continuous sine wave signal*

time = np.linspace(0, 1, 1000)  *# Time range from 0 to 1 second*

*# Define the number of samples, sampling frequency, and parameters for the*

*# discrete sine wave signal*

num\_samples = 100  *# Number of samples*

sampling\_frequency = 10  *# Sampling frequency in Hz*

amplitude = 1  *# Amplitude of the sine wave*

frequency = 2  *# Frequency of the sine wave in Hz*

phase = 0  *# Phase angle of the sine wave in radians*

*# Simulate the continuous sine wave signal*

continuous\_sine\_wave = simulate\_continuous\_sine\_wave(

    time, amplitude, frequency, phase)

*# Simulate the discrete sine wave signal*

discrete\_sine\_wave = simulate\_discrete\_sine\_wave(

    num\_samples, sampling\_frequency, amplitude, frequency, phase)

*# Plot and display the continuous and discrete sine wave signals*

plt.figure(*figsize*=(10, 6))

plt.subplot(2, 1, 1)

plt.plot(time, continuous\_sine\_wave)

plt.title('Continuous Sine Wave Signal')

plt.xlabel('Time (s)')

plt.ylabel('Amplitude')

plt.subplot(2, 1, 2)

plt.stem(discrete\_sine\_wave)

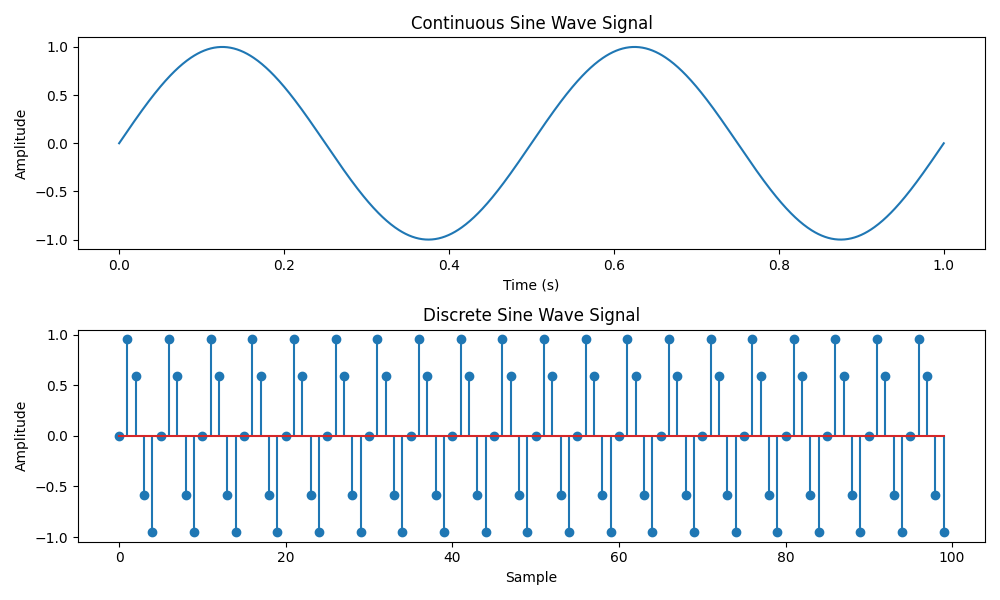
plt.title('Discrete Sine Wave Signal')

plt.xlabel('Sample')

plt.ylabel('Amplitude')

plt.tight\_layout()

plt.show()

**Output:-**

1. Write a Python Program to Simulate y(t) = u(t) + u(t-1) + 3u(t+5).

**Programm:-**

import matplotlib.pyplot as plt

import numpy as np

def simulate\_function(*time*):

    y = np.zeros\_like(time)

    y[time >= 0] = 1

    y[time >= 1] += 1

    y[time >= -5] += 3

    return y

*# Define the time range*

time = np.linspace(-10, 10, 1000)

*# Simulate the function*

function\_values = simulate\_function(time)

*# Plot and display the function*

plt.plot(time, function\_values)

plt.title('Function y(t) = u(t) + u(t-1) + 3\*u(t+5)')

plt.xlabel('Time')

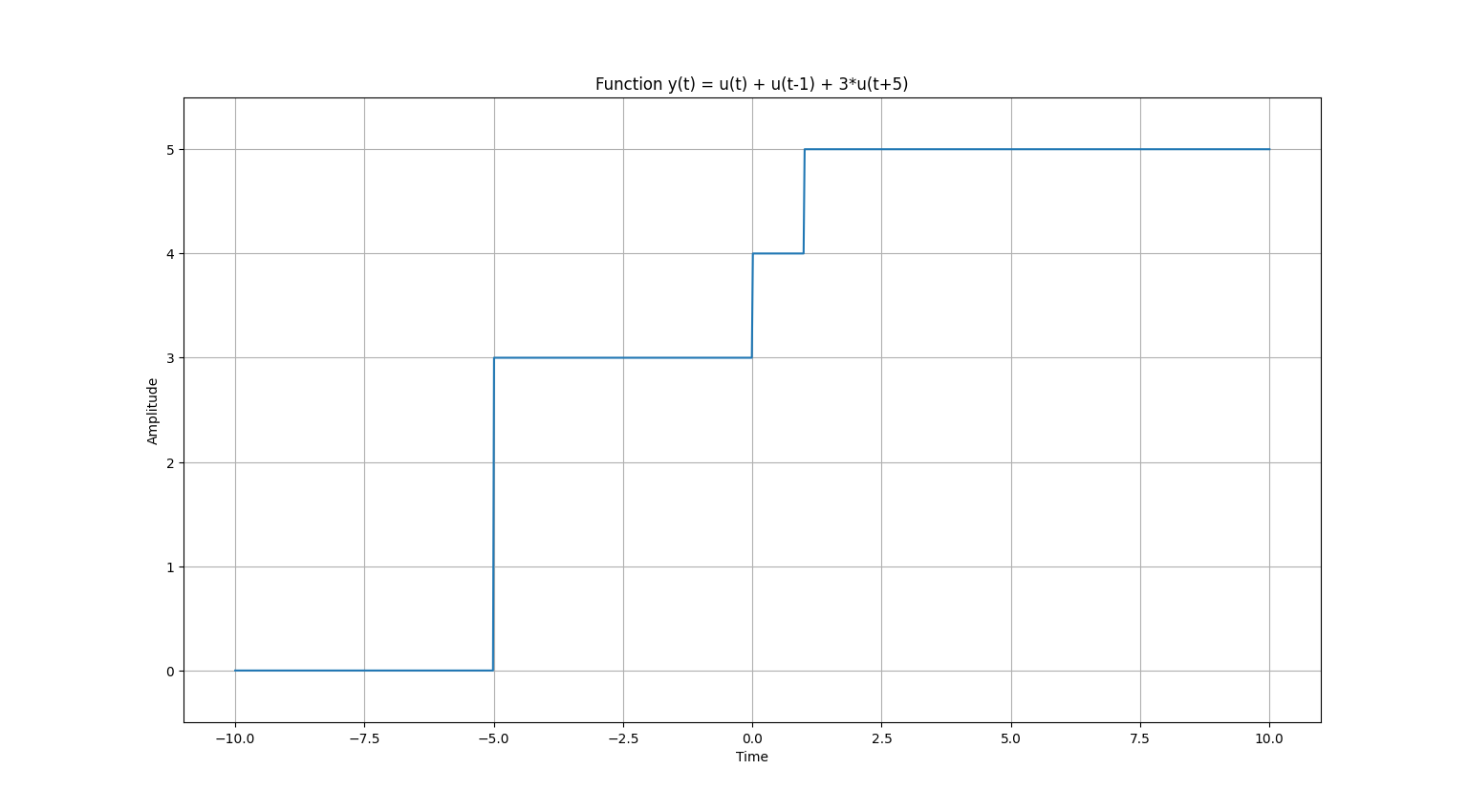
plt.ylabel('Amplitude')

plt.ylim([-0.5, 5.5])

plt.grid(True)

plt.show()

**Output:-**



1. Write a Python Program to Simulate y(t) = Delta(t) + Delta(t-1) + 3\*Delta(t+5).

**Programm:-**

import matplotlib.pyplot as plt

import numpy as np

def simulate\_function(*time*):

    y = np.zeros\_like(time)

    y[time == 0] = 1

    y[time == 1] += 1

    y[time == -5] += 3

    return y

*# Define the time range*

time = np.arange(-10, 11)

*# Simulate the function*

function\_values = simulate\_function(time)

*# Plot and display the function*

plt.stem(time, function\_values)

plt.title('Function y(t) = Delta(t) + delta(t-1) + 3\*delta(t+5)')

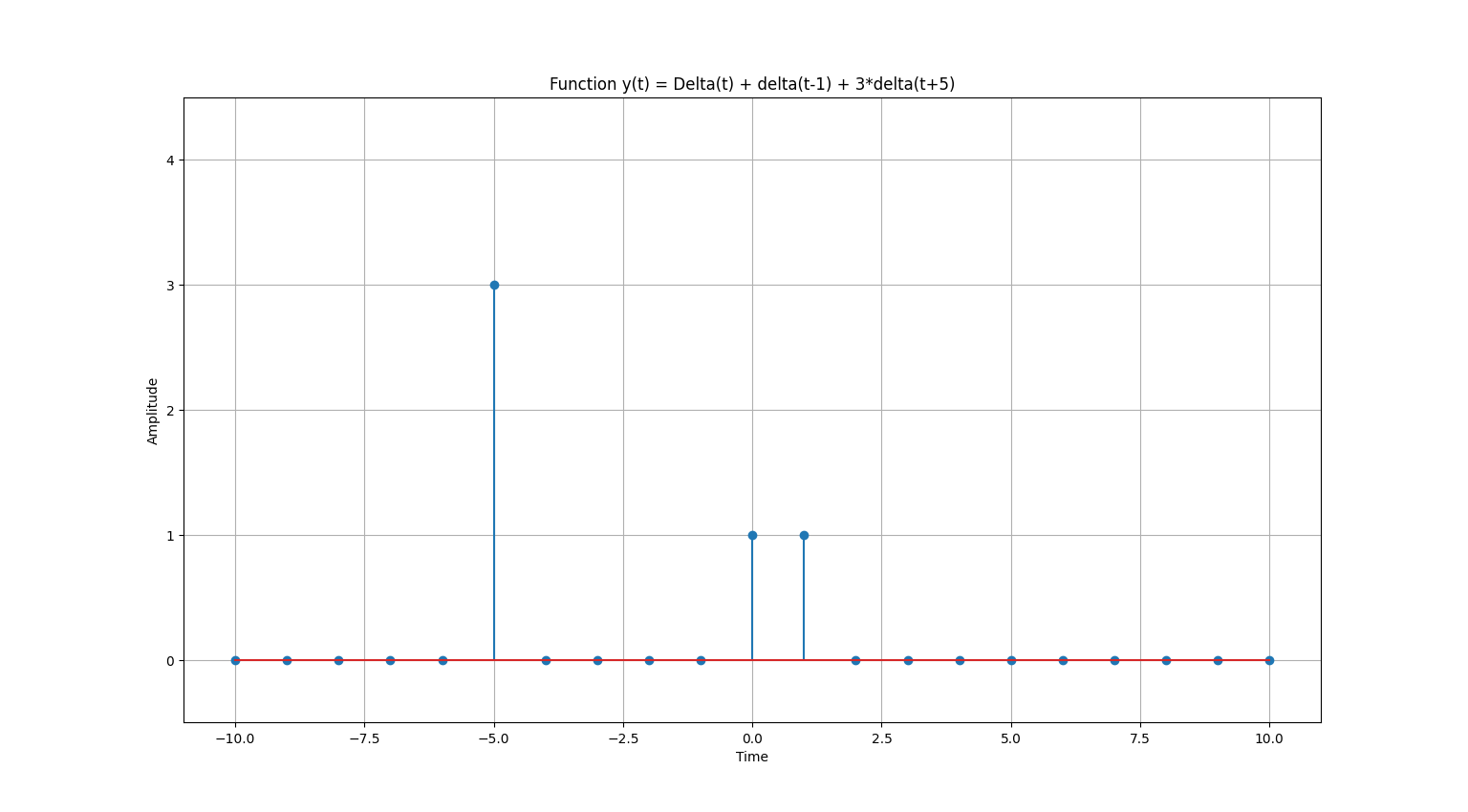
plt.xlabel('Time')

plt.ylabel('Amplitude')

plt.ylim([-0.5, 4.5])

plt.grid(True)

plt.show()

**Output:-**

* **Exercise**.

1. Write a Python Program to Simulate y(t) = Delta(t).

**Programm:-**

import matplotlib.pyplot as plt

import numpy as np

def unit\_impulse(*length*, *position*):

    signal = np.zeros(length)

    signal[position] = 1

    return signal

*# Parameters*

start = -10  *# Start value of the x-axis range*

stop = 10  *# Stop value of the x-axis range*

step = 1  *# Step size*

*# Generate x-axis values*

x = np.arange(start, stop+step, step)

*# Generate unit impulse signal*

impulse\_signal = unit\_impulse(len(x), abs(start)//step)

*# Plot the signal*

plt.stem(x, impulse\_signal)

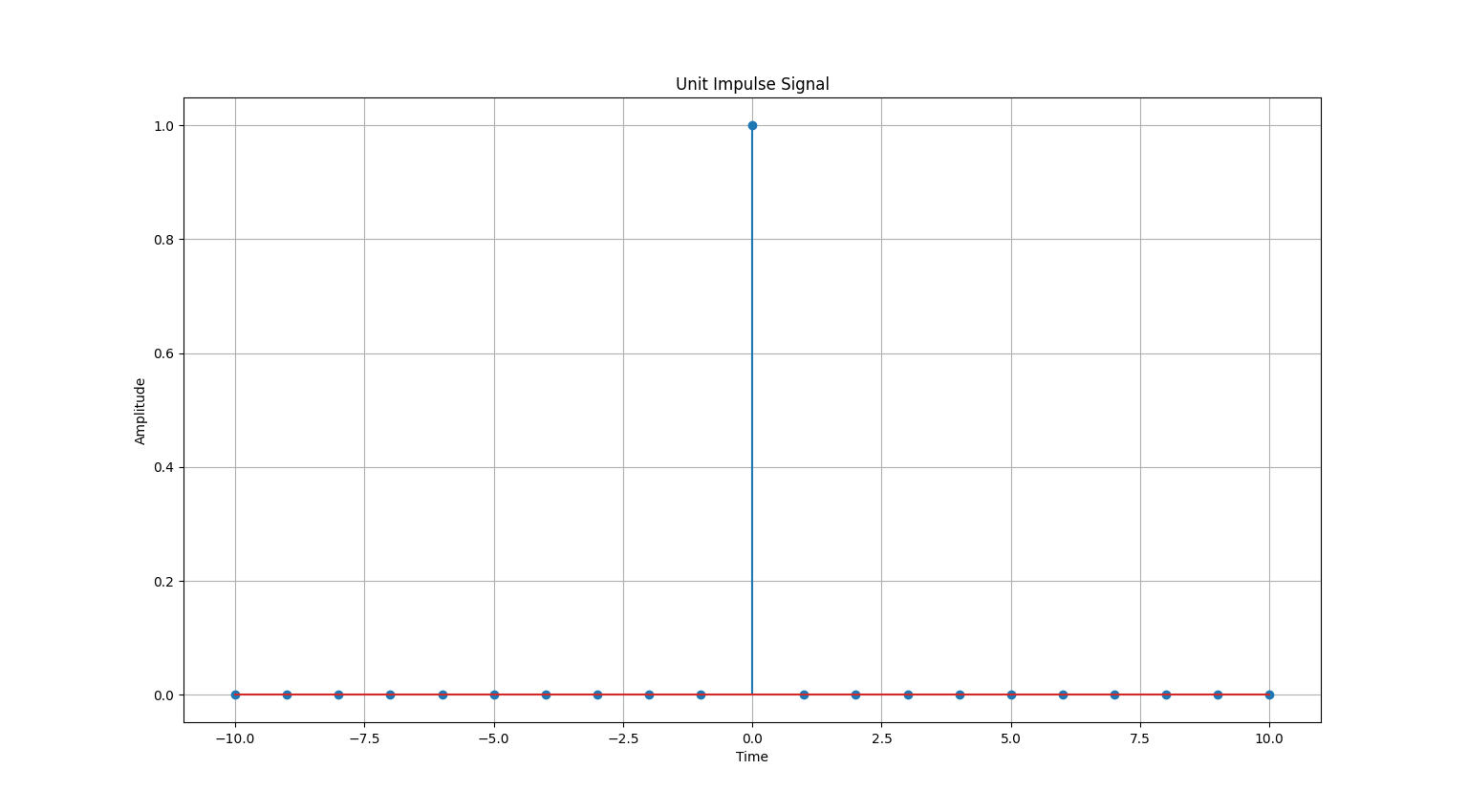
plt.xlabel('Time')

plt.ylabel('Amplitude')

plt.title('Unit Impulse Signal')

plt.grid(True)

plt.show()

**Output:-**

1. Write a Python Program to Simulate y(t) = 3 \* Delta(n) + 5 \* Delta(-n-5) + 8 \* Delta(n-7)

**Programm:-**

import matplotlib.pyplot as plt

import numpy as np

def simulate\_function(*time*):

    y = np.zeros\_like(time)

    y[time == 0] = 3

    y[time == 5] += 5

    y[time == 7] += 8

    return y

*# Define the time range*

time = np.arange(-10, 11)

*# Simulate the function*

function\_values = simulate\_function(time)

*# Plot and display the function*

plt.stem(time, function\_values)

plt.title('Function y(t) = 3 \* delta(n) + 5 \* delta(-n-5) + 8 \* delta(n-7)')

plt.xlabel('Time')

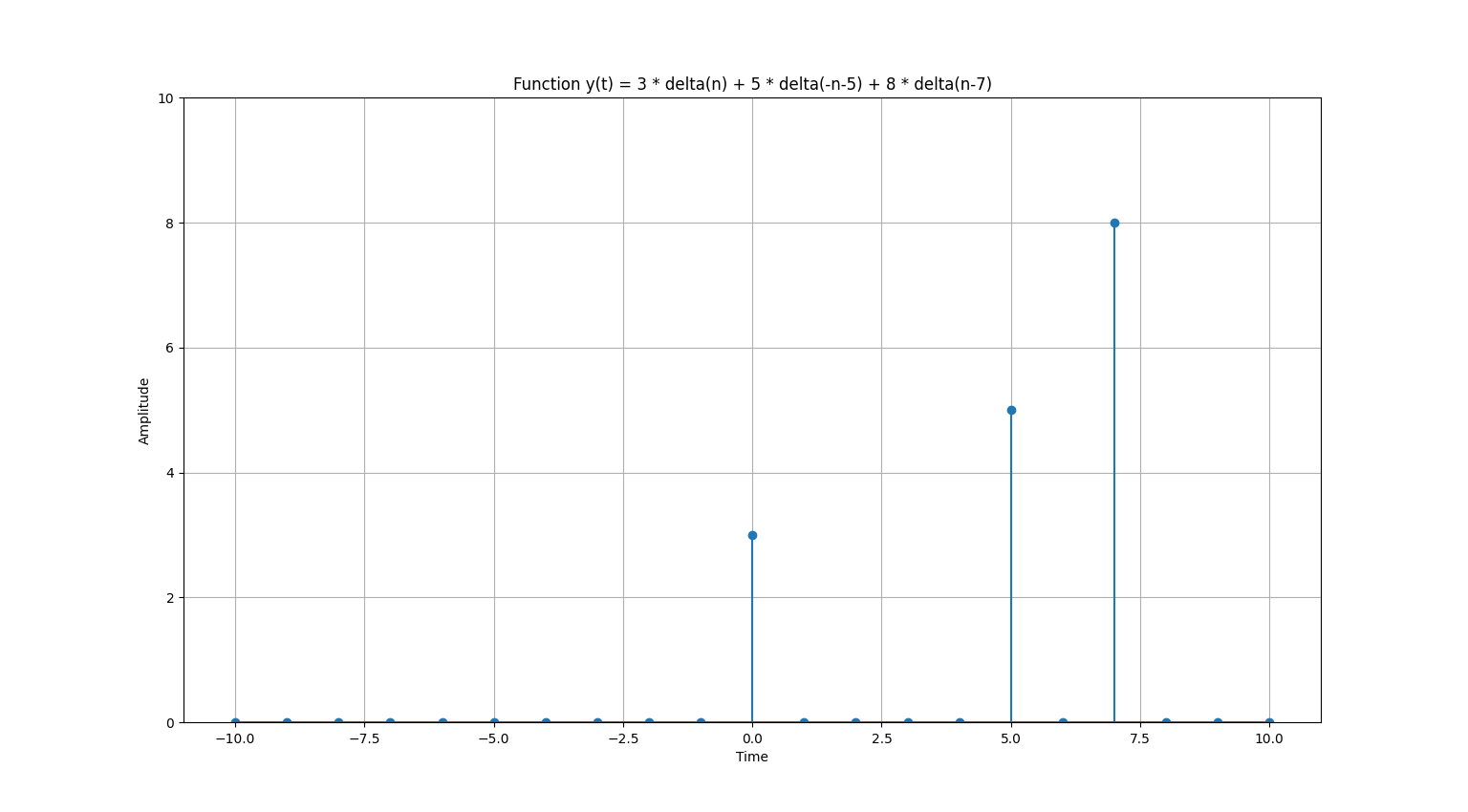
plt.ylabel('Amplitude')

plt.ylim([0, 10])

plt.grid(True)

plt.show()

**Output:-**



1. Write a Python Program to Simulate y(n) = u(n)

**Programm:-**

import matplotlib.pyplot as plt

import numpy as np

def simulate\_discrete\_unit\_step(*num\_samples*):

    unit\_step = np.zeros(num\_samples)

    unit\_step[num\_samples // 2:] = 1

    return unit\_step

*# Define the number of samples for the discrete unit step signal*

num\_samples = 20  *# Number of samples*

*# Simulate the discrete unit step signal*

discrete\_unit\_step = simulate\_discrete\_unit\_step(num\_samples)

*# Plot and display the discrete unit step signal*

plt.figure(*figsize*=(10, 6))

plt.stem(range(-num\_samples//2, num\_samples//2), discrete\_unit\_step)

plt.title('Discrete Unit Step Function u(n)')

plt.xlabel('Sample')

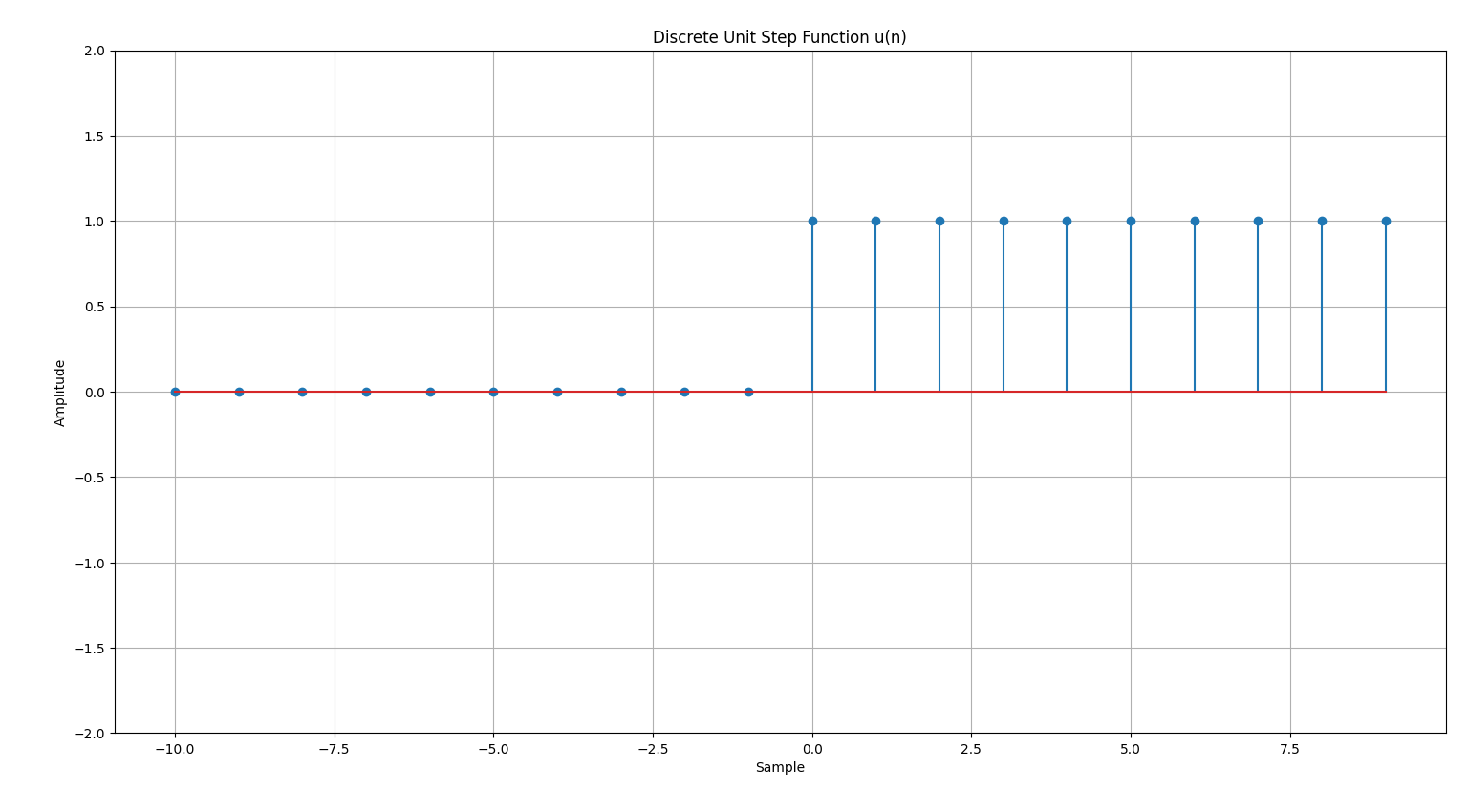
plt.ylabel('Amplitude')

plt.ylim([-2, 2])

plt.grid(True)

plt.tight\_layout()

plt.show()

**Output:-**

1. Write a Python Program to Simulate y(n) =  u(n-7)

**Programm:-**

import matplotlib.pyplot as plt

import numpy as np

def simulate\_discrete\_unit\_step(*num\_samples*, *delay*):

    unit\_step = np.zeros(num\_samples)

    unit\_step[delay:] = 1

    return unit\_step

*# Define the number of samples for the discrete unit step signal*

num\_samples = 20  *# Number of samples*

delay = 7  *# Delay by 7 samples*

*# Simulate the discrete unit step signal*

discrete\_unit\_step = simulate\_discrete\_unit\_step(num\_samples, delay)

*# Plot and display the discrete unit step signal*

plt.figure(*figsize*=(10, 6))

plt.stem(range(num\_samples), discrete\_unit\_step)

plt.title('y(n) = u(n-7)')

plt.xlabel('Sample')

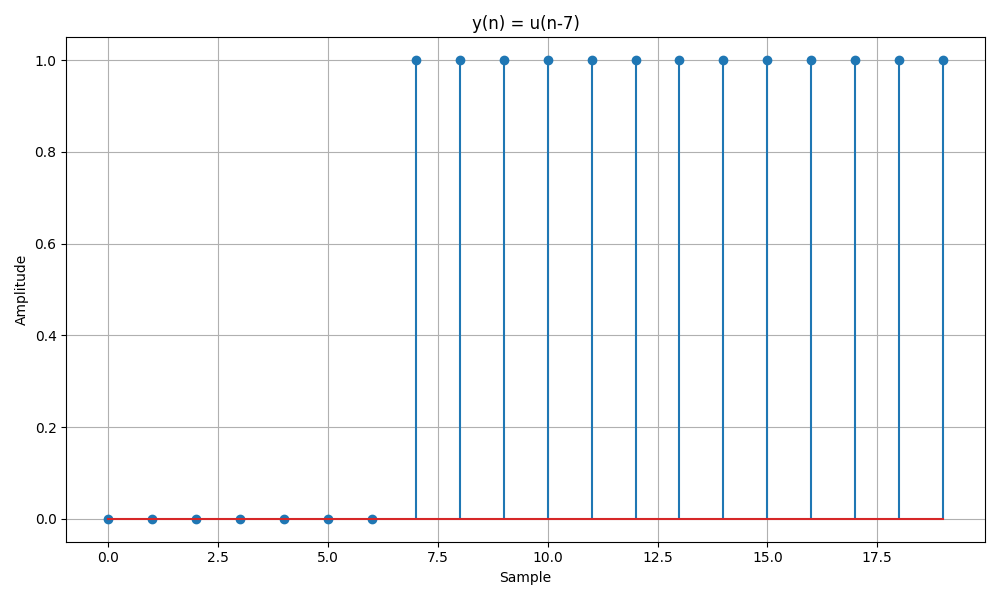
plt.ylabel('Amplitude')

plt.grid(True)

plt.tight\_layout()

plt.show()

**Output:-**



1. Write a Python Program to Simulate y(n) = u(n) + u(n-3) + 6 \* u(n-2) + 8 \* (-n-1)

**Programm:-**

import matplotlib.pyplot as plt

import numpy as np

def simulate\_function(*time*):

    y = np.zeros\_like(time)

    y[time >= 0] = 1  *# u(t)*

    y[time >= 3] += 1  *# u(t-3)*

    y[time >= 2] += 6  *# 6\*u(n-2)*

    y[time <= 1] += 8  *# 8\*u(-n-1)*

    return y

*# Define the time range*

time = np.arange(-10, 11)

*# Simulate the function*

function\_values = simulate\_function(time)

*# Plot and display the function*

plt.stem(time, function\_values)

plt.title('Function y(t) = u(n) + u(n-3) + 6\*u(n-2) + 8\*(-n-1)')

plt.xlabel('Time')

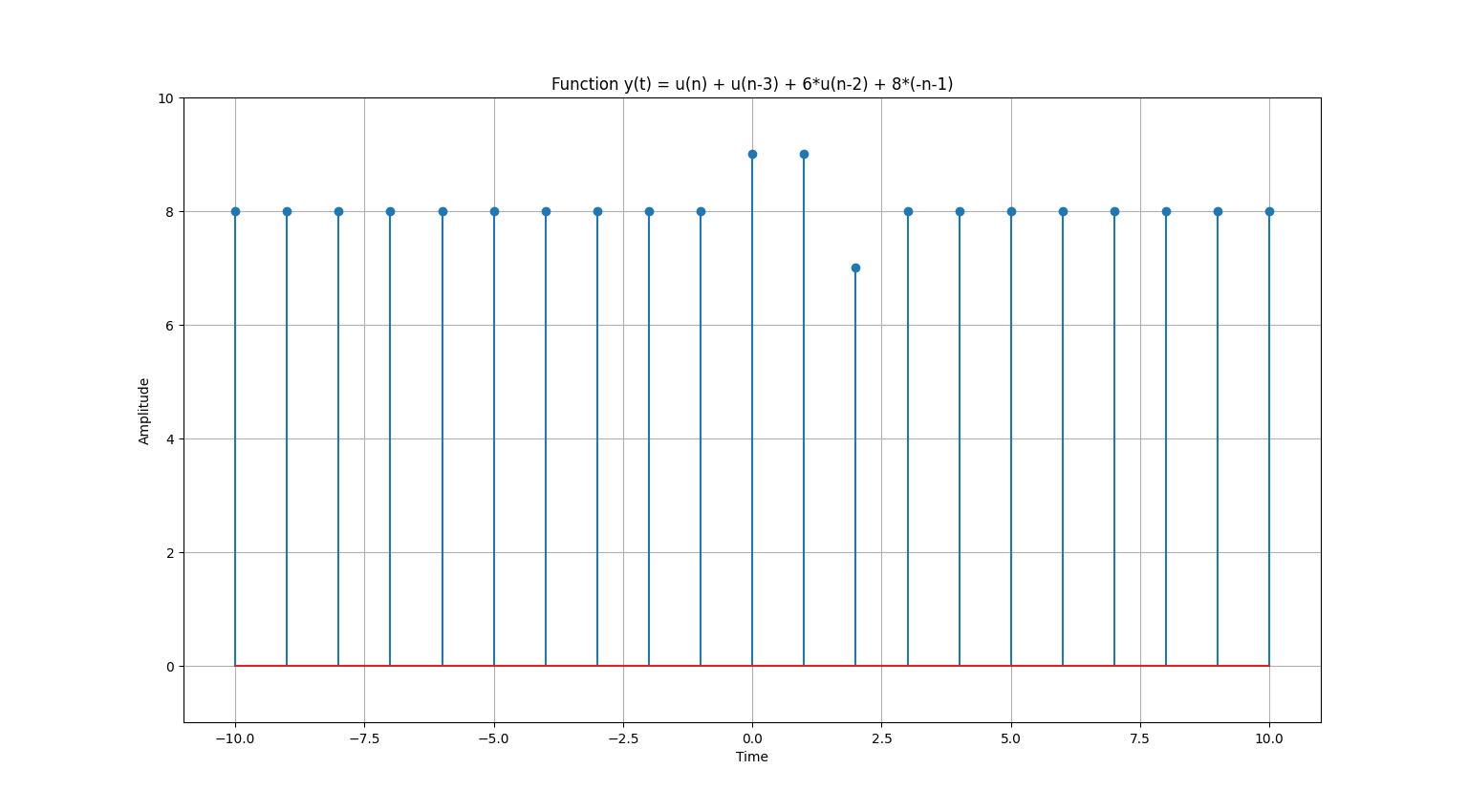
plt.ylabel('Amplitude')

plt.ylim([-1, 10])

plt.grid(True)

plt.show()

**Output:-**



1. Write a Python Program to Simulate y(t) = u(t) + u(t-1) + 3 \* u(t+5)

**Programm:-**

import matplotlib.pyplot as plt

import numpy as np

def simulate\_function(*time*):

    y = np.zeros\_like(time)

    y[time >= 0] = 1

    y[time >= 1] += 1

    y[time >= -5] += 3

    return y

*# Define the time range*

time = np.linspace(-10, 10, 1000)

*# Simulate the function*

function\_values = simulate\_function(time)

*# Plot and display the function*

plt.plot(time, function\_values)

plt.title('Function y(t) = u(t) + u(t-1) + 3\*u(t+5)')

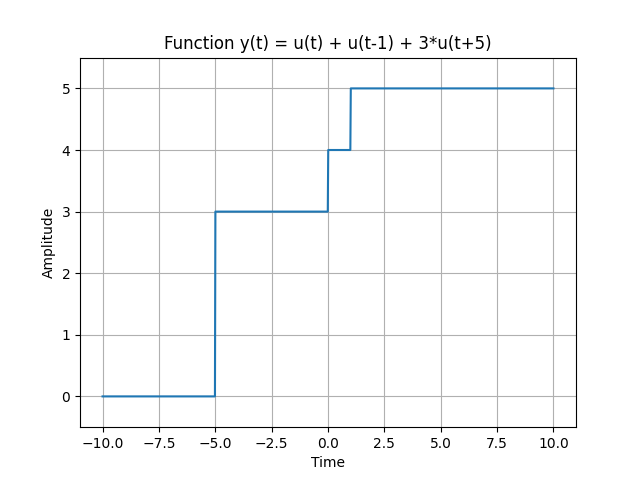
plt.xlabel('Time')

plt.ylabel('Amplitude')

plt.ylim([-0.5, 5.5])

plt.grid(True)

plt.show()

**Output:-**

**Conclusion:-**