Marwadi University	Marwari University Faculty of Technology Department of Information and Communication Technology	
Subject: Digital Signal and Image Processing(01CT0513)	**Aim:** Simulate Discrete Time Sequences.	
Experiment No: 01	**Date: 05-08-2024**	**Enrollment No: 92200133030**

Aim: Simulate Discrete Time Sequences.

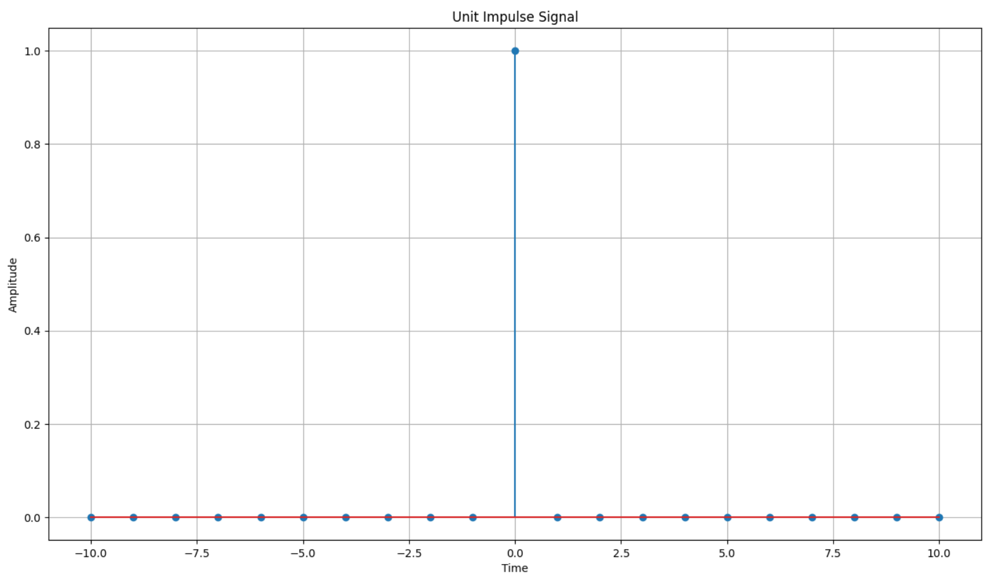
❖ **Write A Python Program to Generate the Given Signals and Plot them using Numpy and Matplotlib.**

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:-



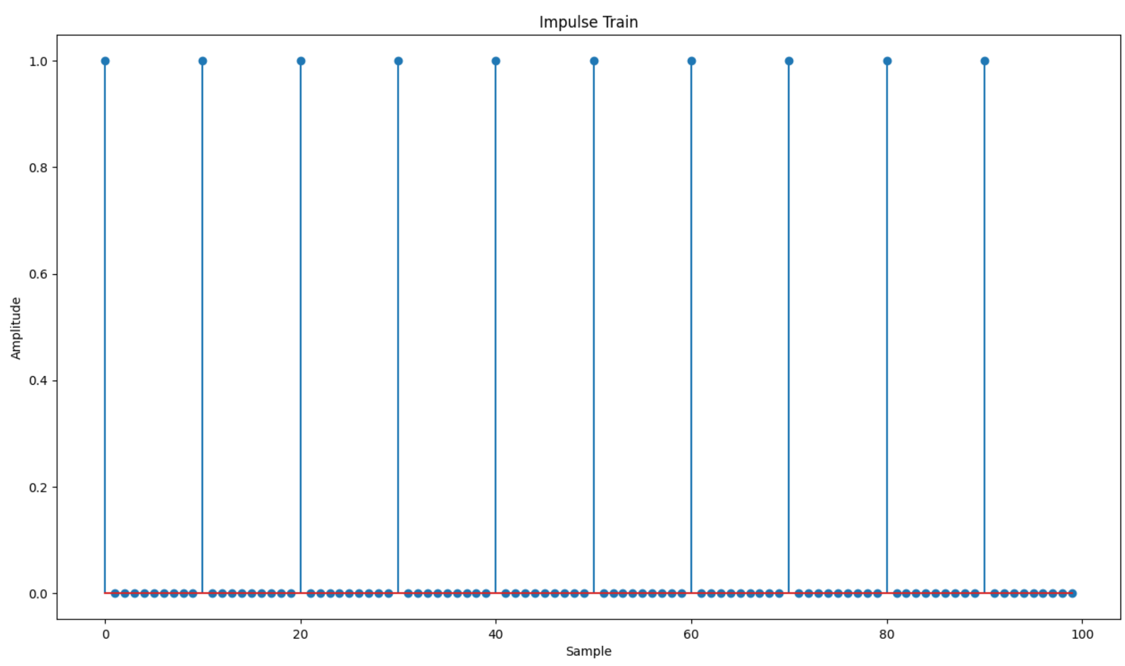
2) Write a Python Programm To Plot Unit Impulse Train.

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:-



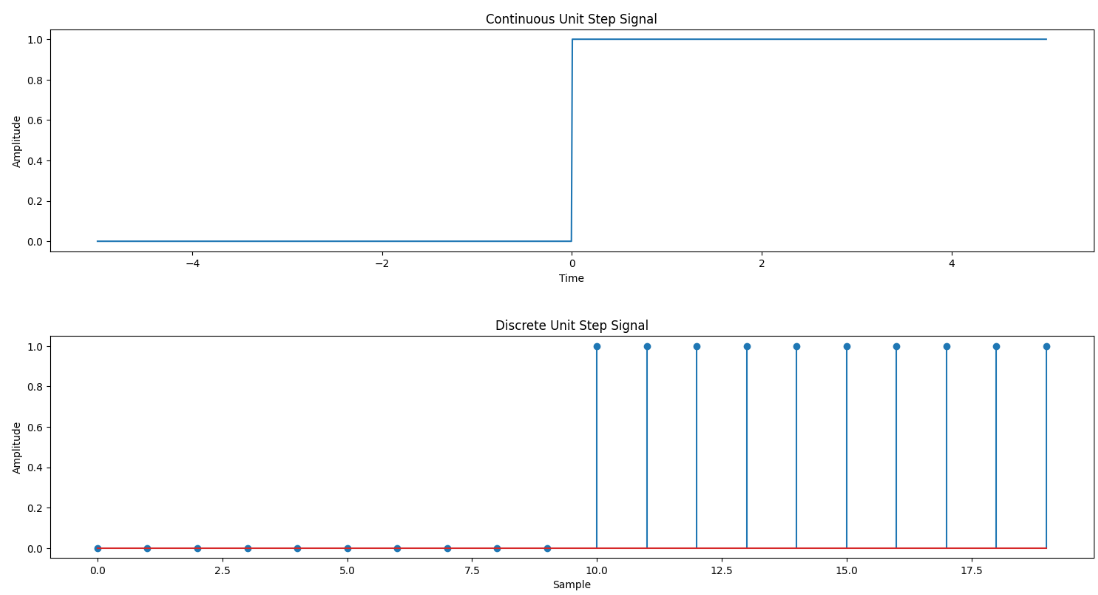
3) Write a Python Program to Simulate Continuous and Discrete Unit Step Signals.

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:-



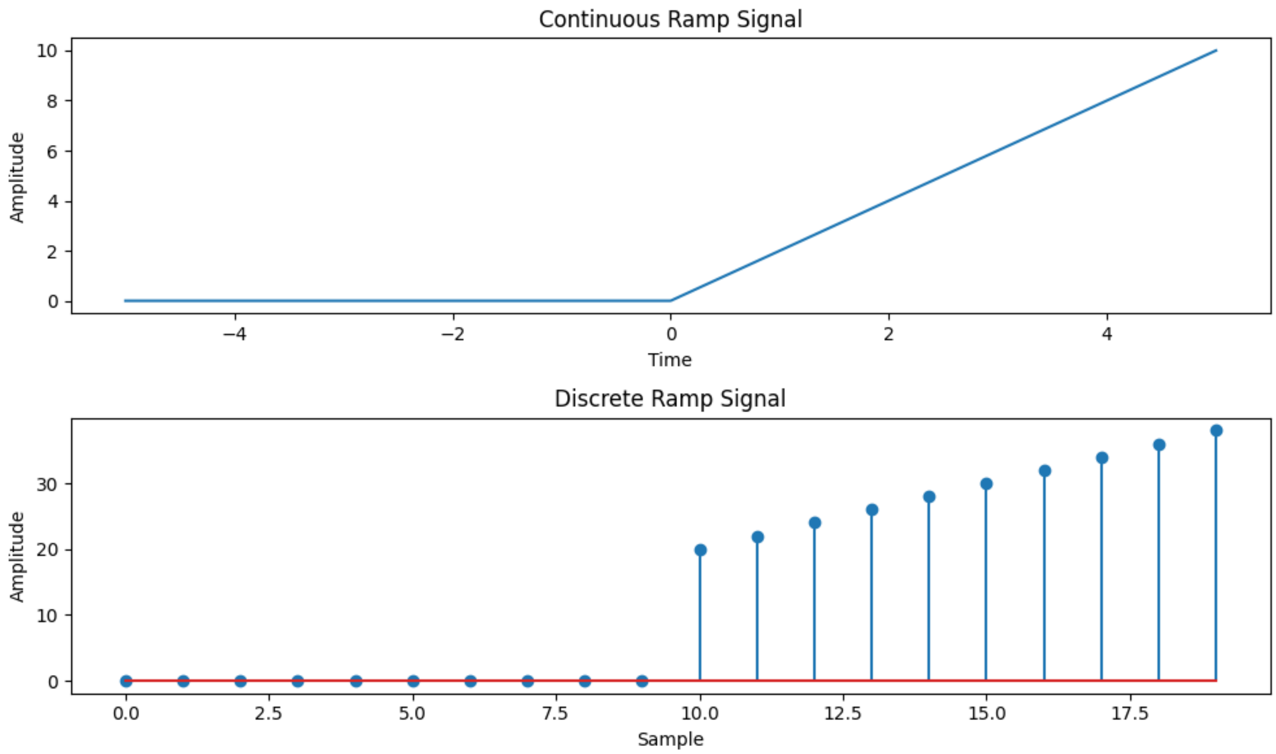
4) Write a Python Program to Simulate Continuous and Discrete Unit Ramp Signals.

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()
```

<u>Output:-</u>



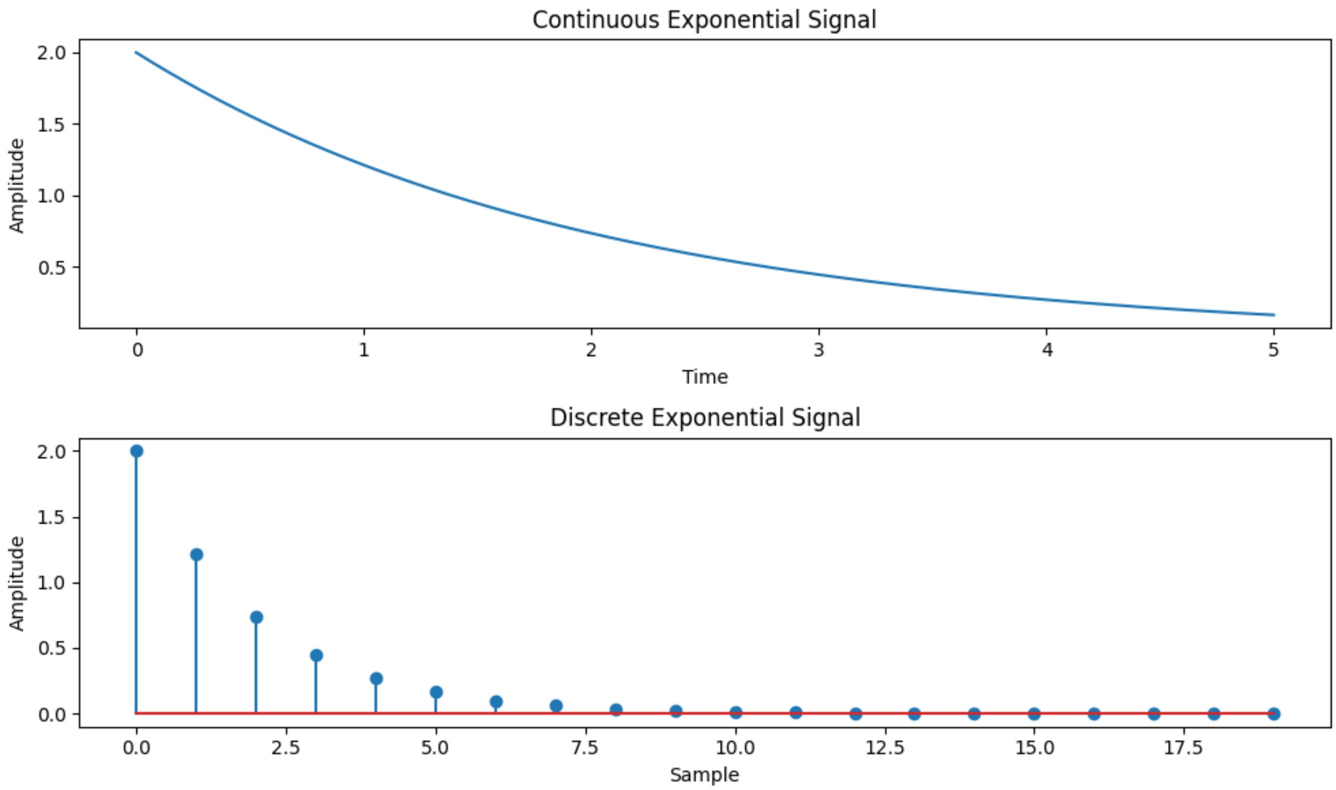
5) Write a Python Program to Simulate Continuous and Discrete Exponential Signals.

<u>Programm:-</u>

```
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:-



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

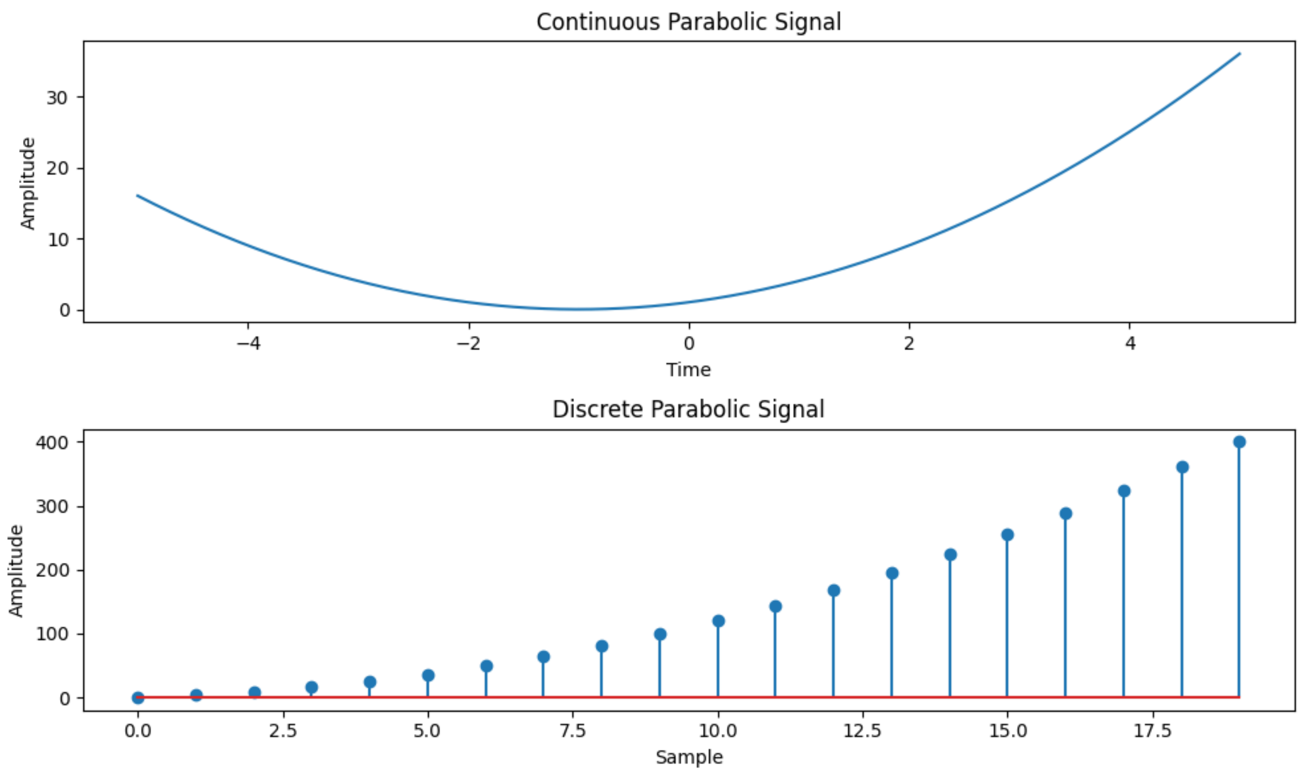
Programm:-

```
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()
```

Marwadi University	Marwari University Faculty of Technology Department of Information and Communication Technology	
Subject: Digital Signal and Image Processing(01CT0513)	**Aim:** Simulate Discrete Time Sequences.	
Experiment No: 01	**Date: 05-08-2024**	**Enrollment No: 92200133030**

Output:-



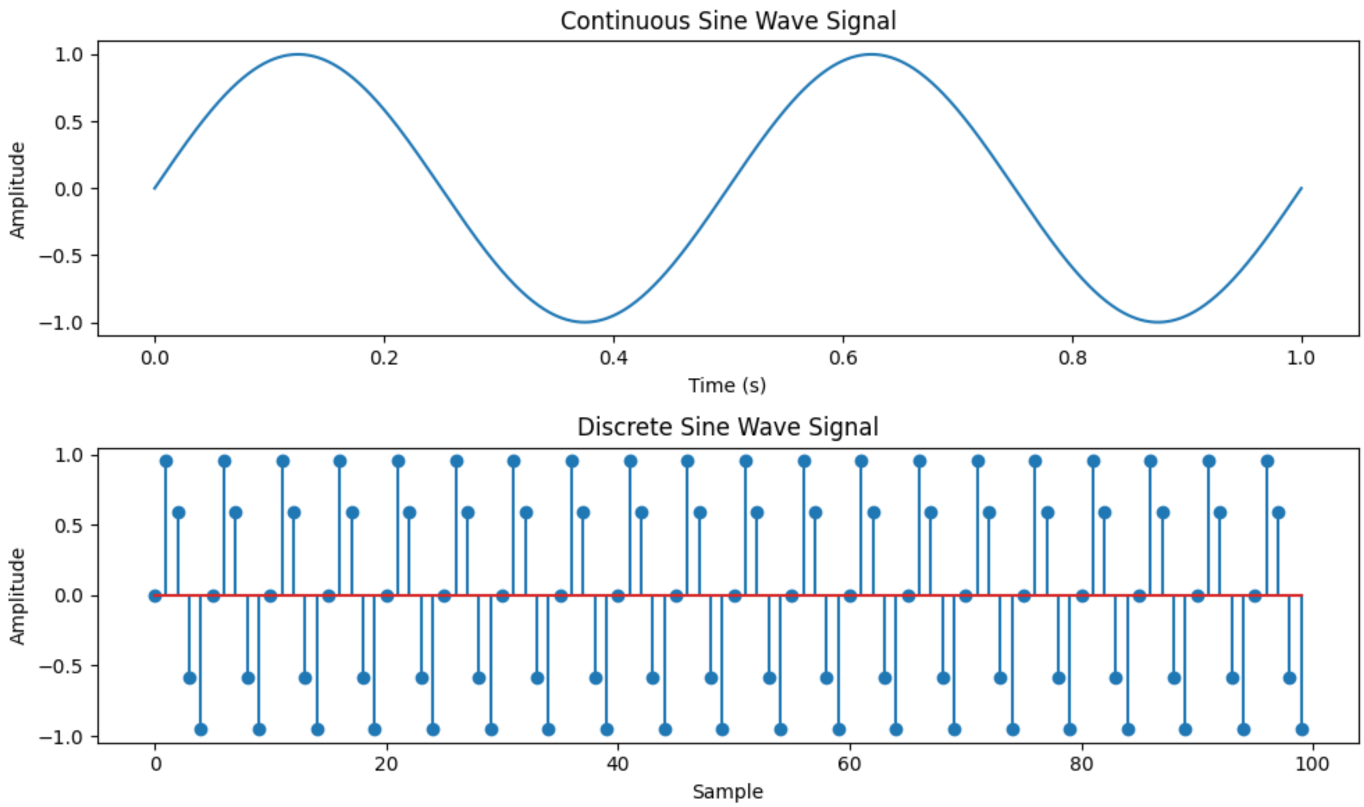
7) Write a Python Program to Simulate Continuous and Discrete Sine Wave Signals.

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()
```

<u>Output:-</u>



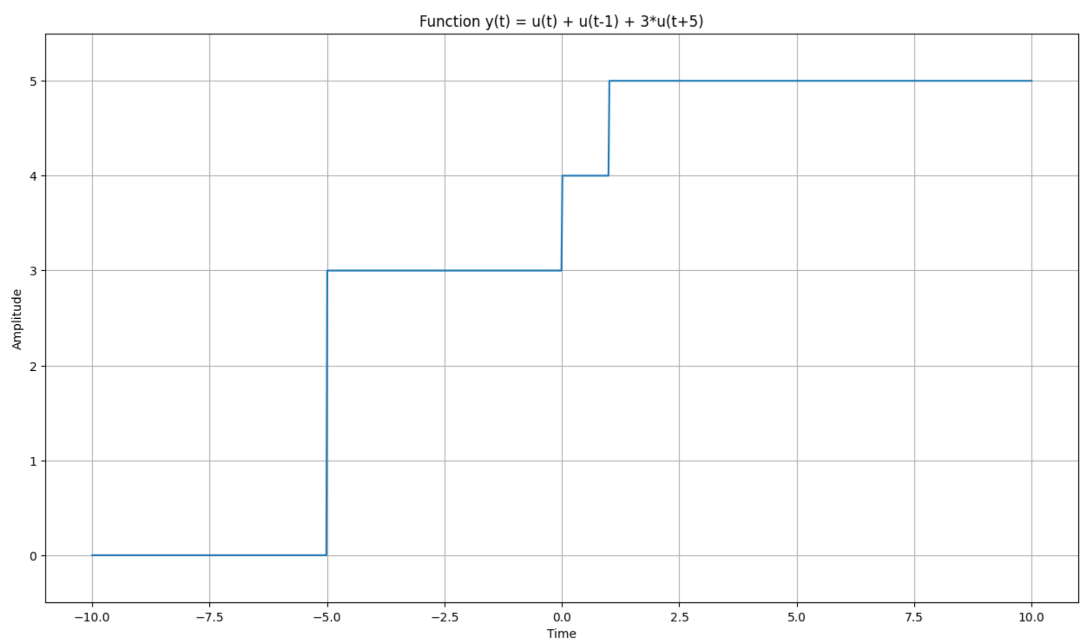
8) Write a Python Program to Simulate y(t) = u(t) + u(t-1) + 3u(t+5).

<u>Programm:-</u>

```
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:-

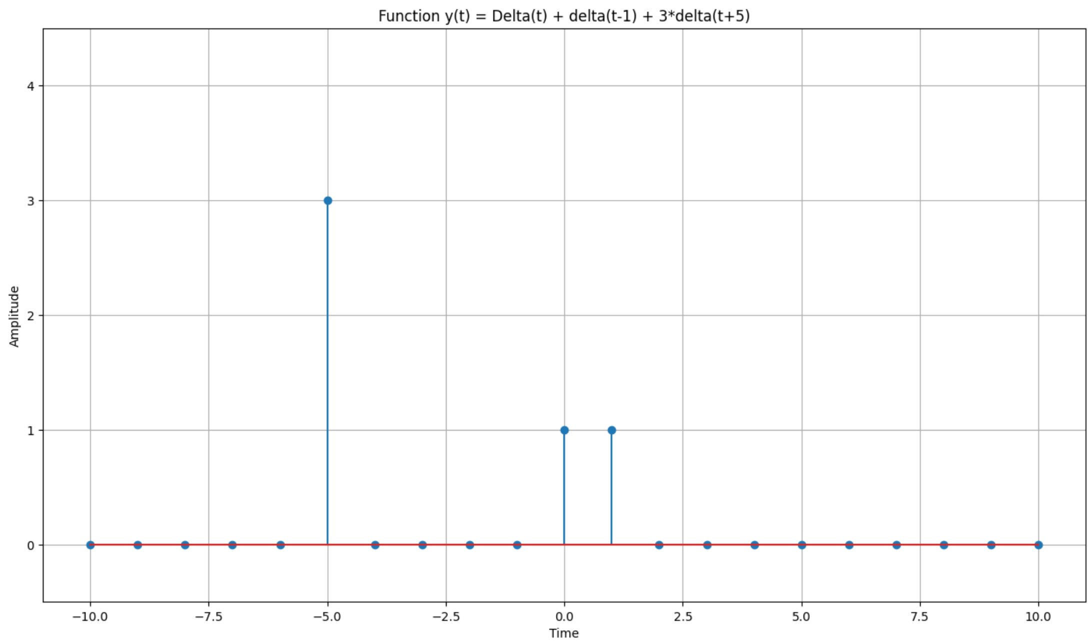


9) Write a Python Program to Simulate y(t) = Delta(t) + Delta(t-1) + 3*Delta(t+5).

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**.

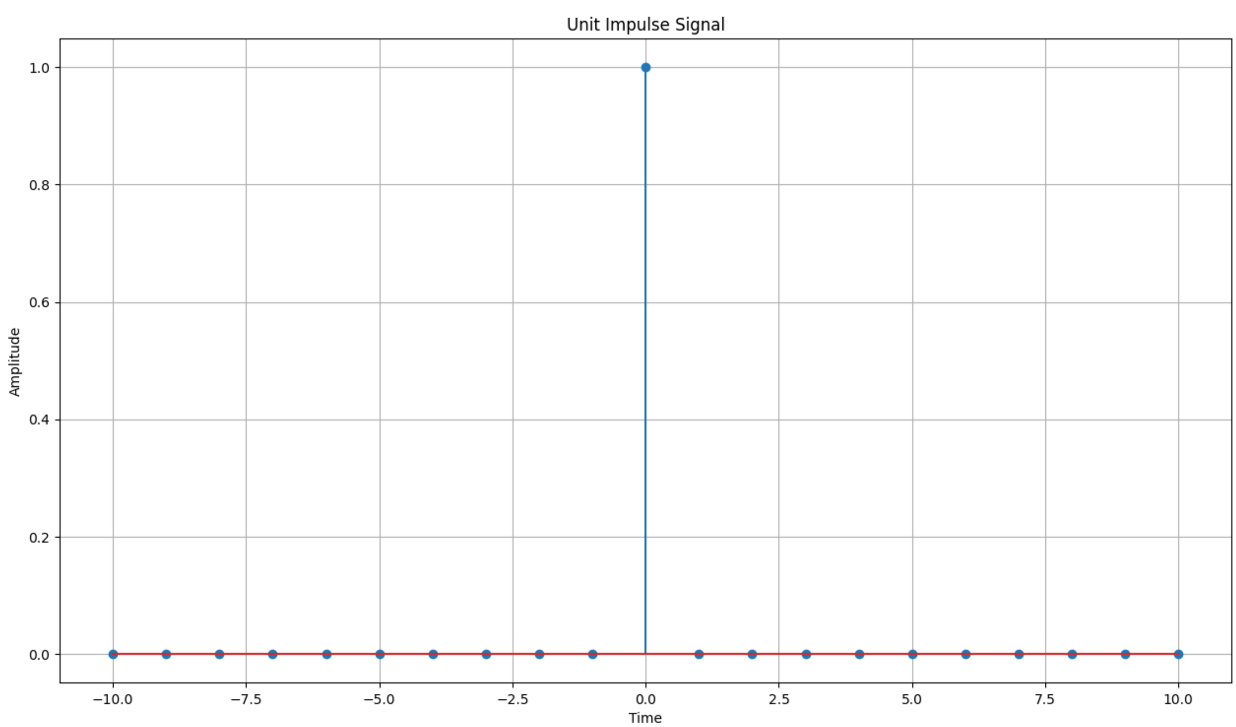
1) Write a Python Program to Simulate y(t) = Delta(t).

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()
```

<u>Output:-</u>

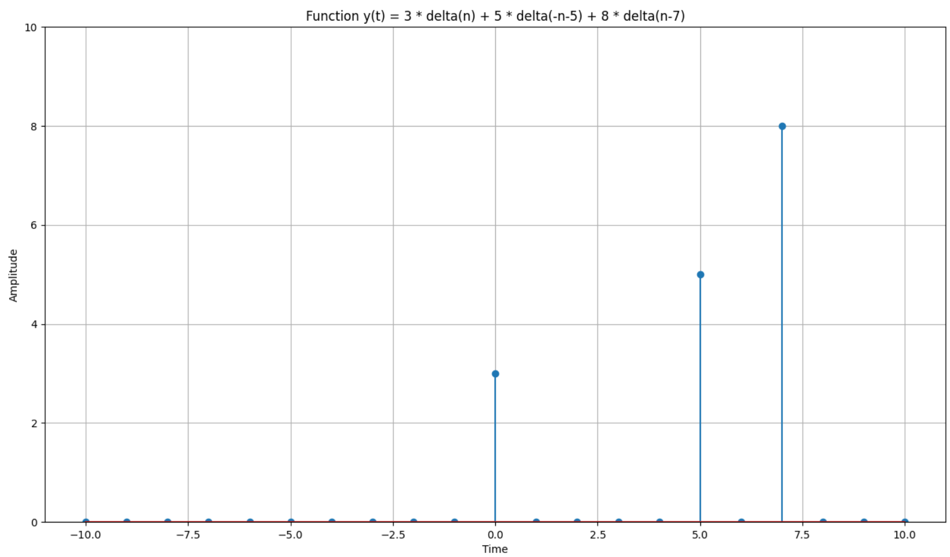


2) Write a Python Program to Simulate y(t) = 3 * Delta(n) + 5 * Delta(-n-5) + 8 * Delta(n-7)

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:-



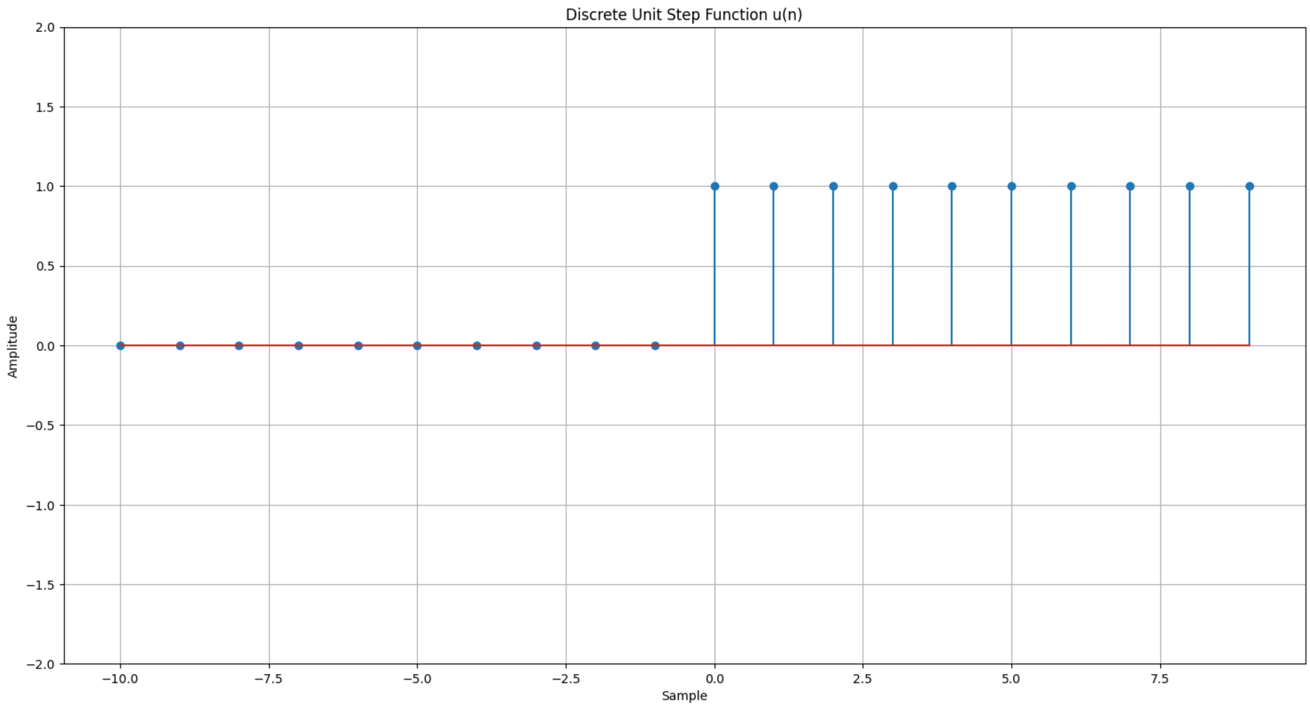
3) Write a Python Program to Simulate y(n) = u(n)

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:-

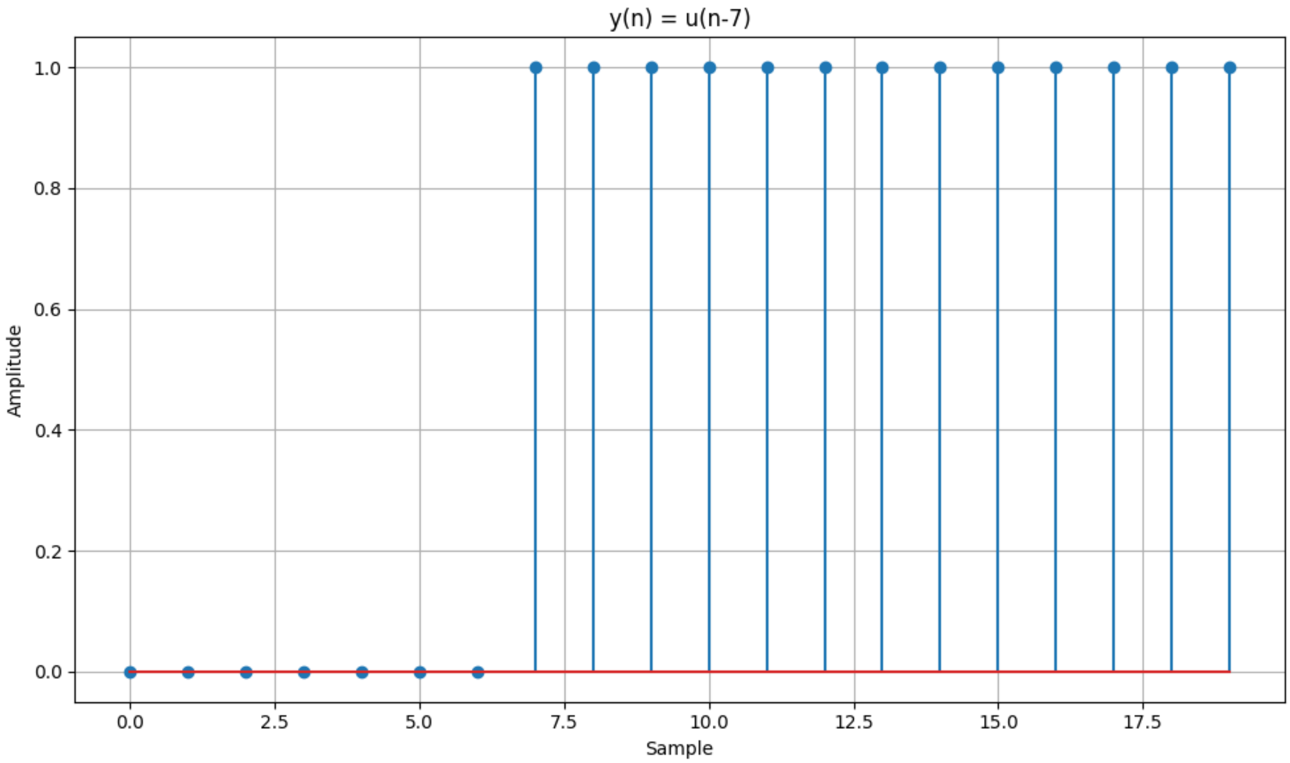


4) Write a Python Program to Simulate y(n) = u(n-7)

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:-



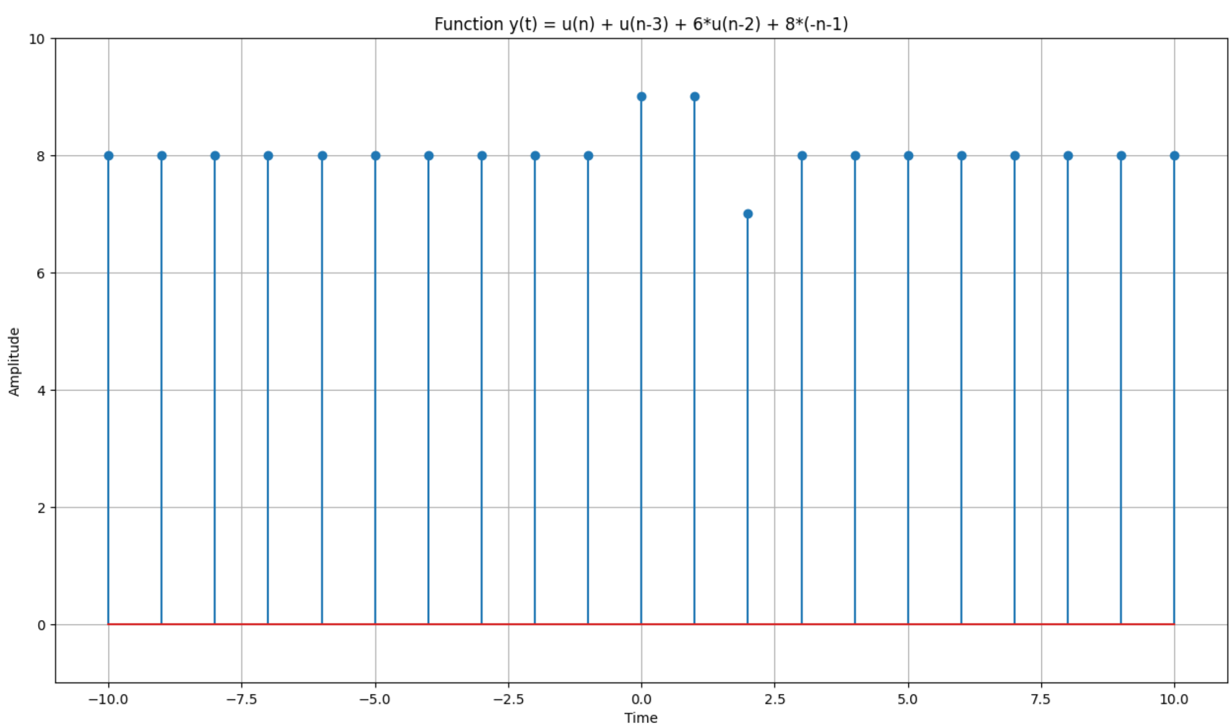
5) Write a Python Program to Simulate y(n) = u(n) + u(n-3) + 6 * u(n-2) + 8 * (-n-1)

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:-

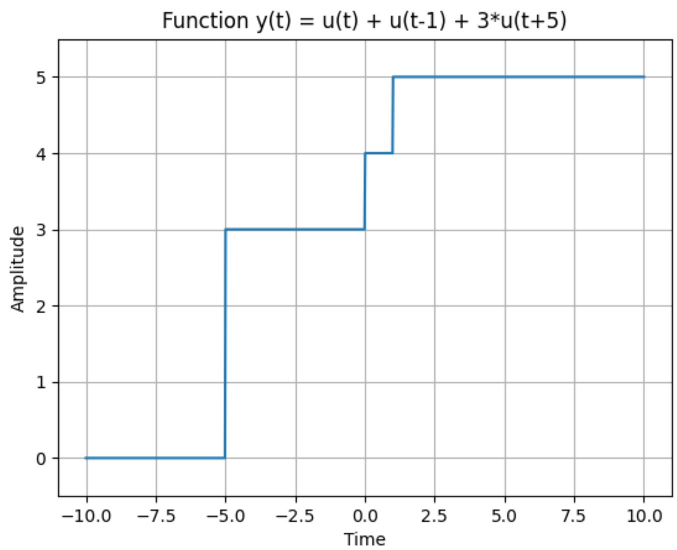


6) Write a Python Program to Simulate y(t) = u(t) + u(t-1) + 3 * u(t+5)

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:-