```
function [x,n,NRate,s1,s2,s3,yq,MSE,Response,I]=fname1
clc; clear all;
% De La Salle University
% Electronics and Computer Engineering Department
%
% Course        : LBYCPA4
% SECTION       : EQ4
% Submitted by  : Jared Craig C. Ong
% Submitted to  : Dr. Lawrence Materum
%
% Exercise 1: Discrete-time Signals
%
% Note: Check the instructions given in the canvas

%% Task 1
% Place your code here to generate x1, x2, x3,x4,x5, and x6
n=-25:25;

% Place your code here to create the visualization of Figure 1

function y = del(n)
    y = zeros(1, length(n));
    [a, b] = find(n == 0);
    y(b) = 1;
end

function y = step(n)
    y = zeros(1, length(n));
    [a, b] = find(n >= 0);
    y(b) = 1;
end

x1 = -del(n + 3) + del(n) + del(n - 5);
x2 = step(n + 3) - step(n - 5);
x3 = (-27/40).^n .* step(n);
x4 = cos(3*pi*n/12 - pi/6);
x5 = (-27/40).^n .* cos(3*pi*n/12 - pi/6) .* step(n);
x6 = cos(3*pi*n/12 - pi/6) .* (step(n + 3) - step(n - 5));

% Create the visualization of Figure 1
figure(1);

% Plot x1[n]
subplot(3, 2, 1);
stem(n, x1, 'filled');
title('x_1[n]');
xlabel('n');
ylabel('x_1[n]');
```

```
% Plot x2[n]
subplot(3, 2, 2);
stem(n, x2, 'filled');
title('x_2[n]');
xlabel('n');
ylabel('x_2[n]');

% Plot x3[n]
subplot(3, 2, 3);
stem(n, x3, 'filled');
title('x_3[n]');
xlabel('n');
ylabel('x_3[n]');

% Plot x4[n]
subplot(3, 2, 4);
stem(n, x4, 'filled');
title('x_4[n]');
xlabel('n');
ylabel('x_4[n]');

% Plot x5[n]
subplot(3, 2, 5);
stem(n, x5, 'filled');
title('x_5[n]');
xlabel('n');
ylabel('x_5[n]');

% Plot x6[n]
subplot(3, 2, 6);
stem(n, x6, 'filled');
title('x_6[n]');
xlabel('n');
ylabel('x_6[n]');

% The following code will concatenate the six discrete-time sequences. x
% represents the disctrete-time sequences with 6 channels.
x=[x1' x2' x3' x4' x5' x6'];


%% Task 2
% Place your code here to define the Nyquist rate using the variable NRate
t = -0.5:0.001:0.5;
s = (0.5 * cos(38*pi*t)) + (0.5 * cos(42*pi*t));

% Place your code here to define Fs1 and generate t1 and s1
sone = 42*5;
int1 = 1/sone;
v1 = -0.5:int1:0.5;
```

```
s1 = (0.5 * cos(38*pi*v1)) + (0.5 * cos(42*pi*v1));

% Place your code here to define Fs2 and generate t2 and s2
stwo = 42;
int2 = 1/stwo;
v2 = -0.5:int2:0.5;
s2 = (0.5 * cos(38*pi*v2)) + (0.5 * cos(42*pi*v2));

% Place your code here to define Fs3 and generate t3 and s3
sthree = 42 * (3/4);
int3 = 1/sthree;
v3 = -0.5:int3:0.5;
s3 = 0.5 * cos(38*pi*v3) + 0.5 * cos(42*pi*v3);

% Place your code here to create visualization of Figure 2

figure;
subplot(3,1,1);
plot(t,s); hold on;
stem(v1,s1,'g');
title('Sampling Rate = 5Nyquist Rate');
xlabel('Time(s)'); ylabel('Amplitude');
legend('continious', 'sampled');

subplot(3,1,2);
plot(t,s); hold on;
stem(v2,s2,'g');
title('Sampling Rate = Nyquist Rate');
xlabel('Time(s)'); ylabel('Amplitude');
legend('continious', 'sampled');

subplot(3,1,3);
plot(t,s); hold on;
stem(v3,s3,'g');
title('Sampling Rate = 3/4Nyquist Rate');
xlabel('Time(s)'); ylabel('Amplitude');
legend('continious', 'sampled');

%% Task 3: Quantization of audio into n number of bits

% Read the audio 'tononoise8mono.wav' and store audio data and sampling rate
[y, Fs] = audioread('tononoise8mono.wav');
% y contains the audio samples, Fs is the sampling frequency

% Store the number of samples to N and generate the corresponding continuous time t
N = length(y);
t = (0:N-1)/Fs;  % Time vector

% Convert the audio data into 8-bit signed integer (normalized from -1 to 1 to 8-bit ↙
```

```
scale)
y8 = 128 *y;
% Initialize an empty matrix to store quantized audio for different bit levels
yq = zeros(N, 8);

% Quantize the audio data into n-bits and store it in yq (column-wise storage)
for nbits = 7:-1:1
    y_n = round((2^(nbits - 1)) * y);  % Quantize to nbits

    yq(:, 8-nbits) = y_n;              % Store quantized audio
end

% Plot the original and quantized audio data in figure 3
figure(3);
plot(t, y, 'k', 'DisplayName', 'Original 8-bit Audio'); hold on;
for nbits = 7:-1:1
    plot(t, yq(:, 8-nbits), 'DisplayName', [num2str(nbits), '-bit Quantized Audio']);
end
xlabel('Time (s)');
ylabel('Amplitude');
legend;
title('Original and Quantized Audio Signals');
hold off;

% Compute the mean square error for each bit depth and store in MSE
MSE = zeros(7, 1);
for nbits = 7:-1:1
    y_n = yq(:, 8-nbits);  % Retrieve the quantized signal
    MSE(nbits) = (1/N) * sum((y_n - y).^2);  % Compute MSE between original and↙
quantized signals
end

% Plot the mean square error in figure 4
figure(4);
plot(7:-1:1, MSE, '-o');
xlabel('Number of Bits');
ylabel('Mean Squared Error (MSE)');
title('MSE vs. Number of Quantization Bits');
grid on;

% Is the MSE consistent with the played back audio? (Yes or No)
Response = 'Yes';

% Play back the audio for the lowest number of bits (e.g., 1-bit quantized audio)
sound(yq(:, 8), Fs);  % Play back 1-bit quantized audio


%% Task 4: Image Generation
% Place your code here to create the image I
```

```
% Initialize I
I = zeros(140,140,3);
colors =[
% Generate white
  1, 1, 1;
% Generate Yellow
 1, 1, 0;
% Generate Cyan
 0, 1, 1;
% Generate Green
0, 1, 0;
% Generate Magenta
1, 0, 1;
% Generate Red
 1, 0, 0;
% Generate Blue
0, 0, 1;
];
% Place your code here to cdisplay of the image in figure 5
% Fill the image matrix with color bars
for i = 1:7
    I(:, (i-1)*(140/7) + 1:i*(140/7), :) = repmat(reshape(colors(i, :), 1, 1, 3), ↙
140, (140/7));
end

% Display the generated image using imagesc to get axes
figure(5);
imagesc([1 140], [1 140], I);  % Use imagesc to show axes and grid

% Set axis properties
title('Generated Color Bars with Axis Values');
axis on;          % Turn on axis


end
```