# Lab 1

## Name : Abhinav Pandey

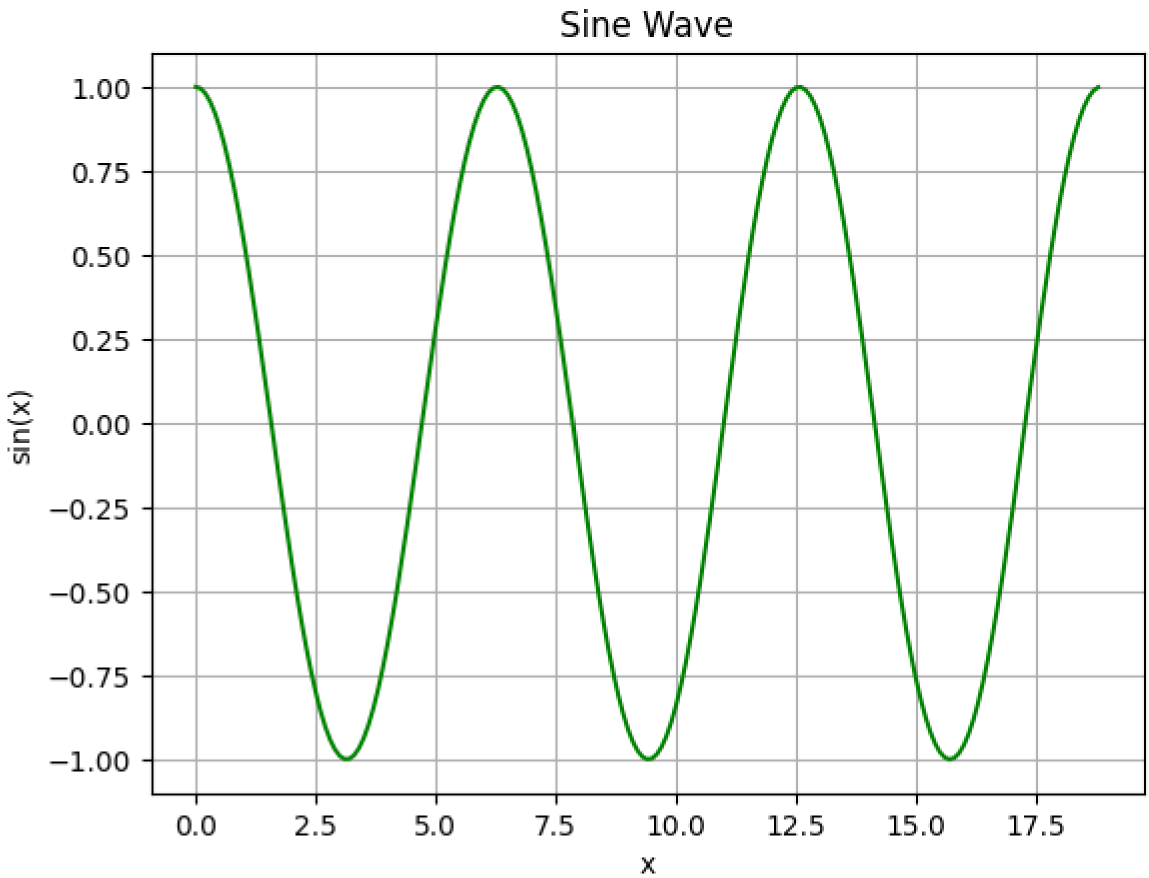
## Roll No : AM.EN.U4AIE21088

1. Try the following Python code to plot sine wave using Matplotlib.

```
#Importing required library

import numpy as np
import matplotlib.pyplot as plt

# Creating x axis with range and y axis with Sine
# Function for Plotting Sine Graph

x = np.arange(0, 6*np.pi, 0.1)
y = np.sin(x)

# Plotting Sine Graph

plt.plot(x, y, color='green')
plt.show()
```

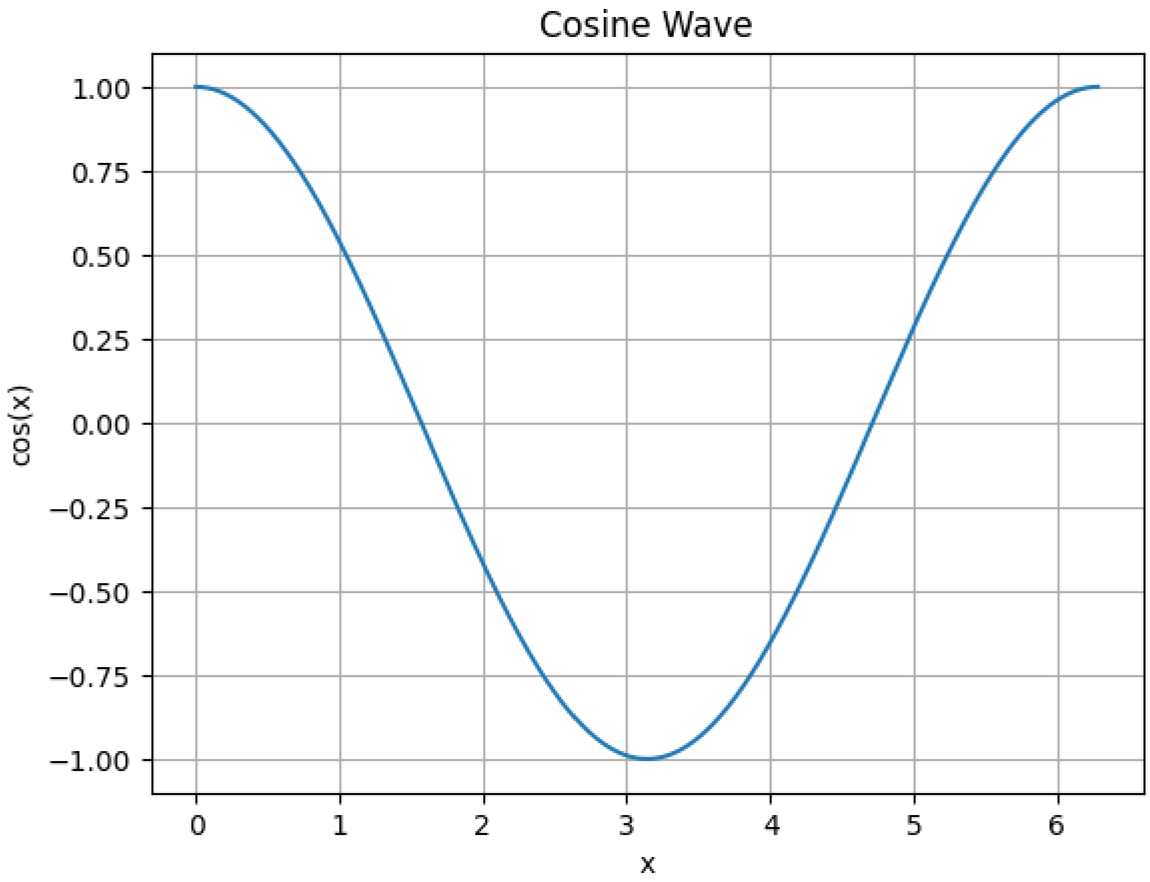
In [ ]:

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

x = np.arange(0,6*np.pi, 0.1)
y = np.cos(x)

plt.plot(x,y,color="green")
plt.title('Sine Wave')
plt.xlabel('x')
plt.ylabel('sin(x)')
plt.grid(True)
plt.show()
```

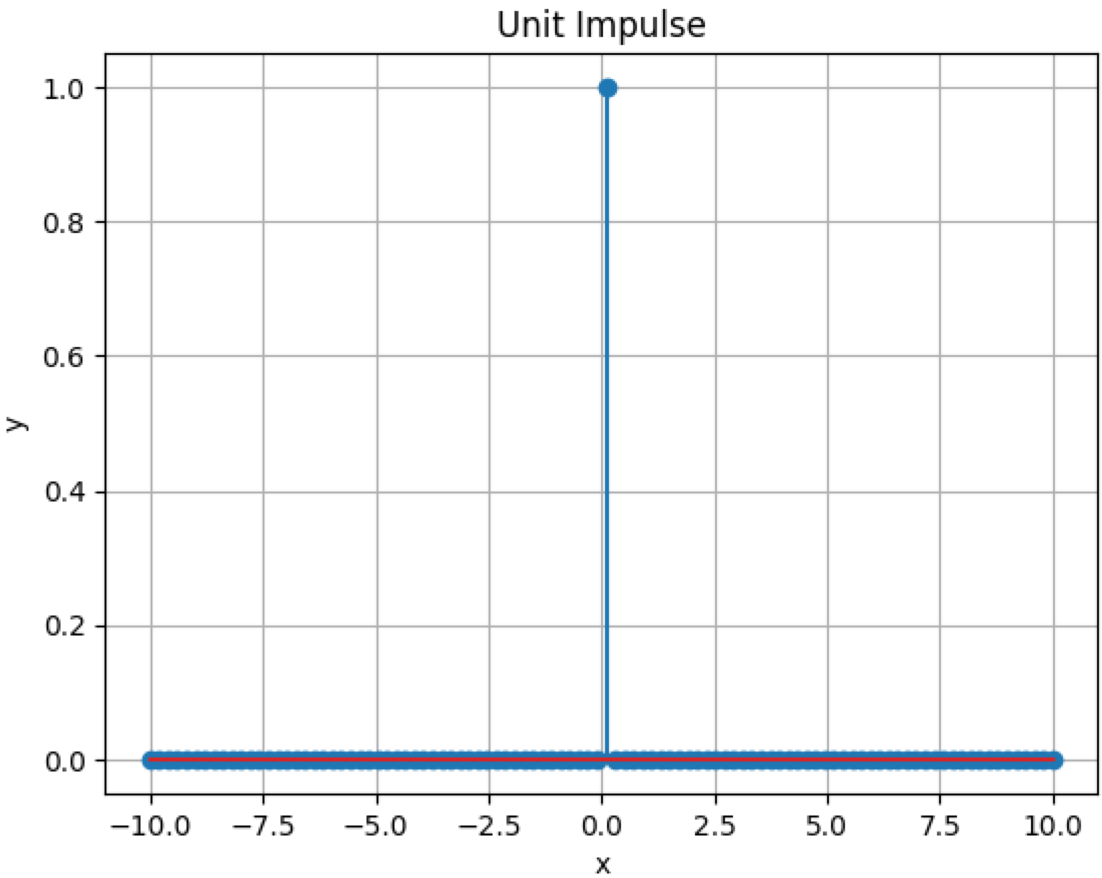



2. Try plotting the following signals in the same way as in Q1 [Hint: You may need SciPy]
   i. Cosine wave
   ii. Unit impulse
   iii. Unit step wave
   iv. Square wave
   v. Exponential waveform
   vi. Sawtooth waveform

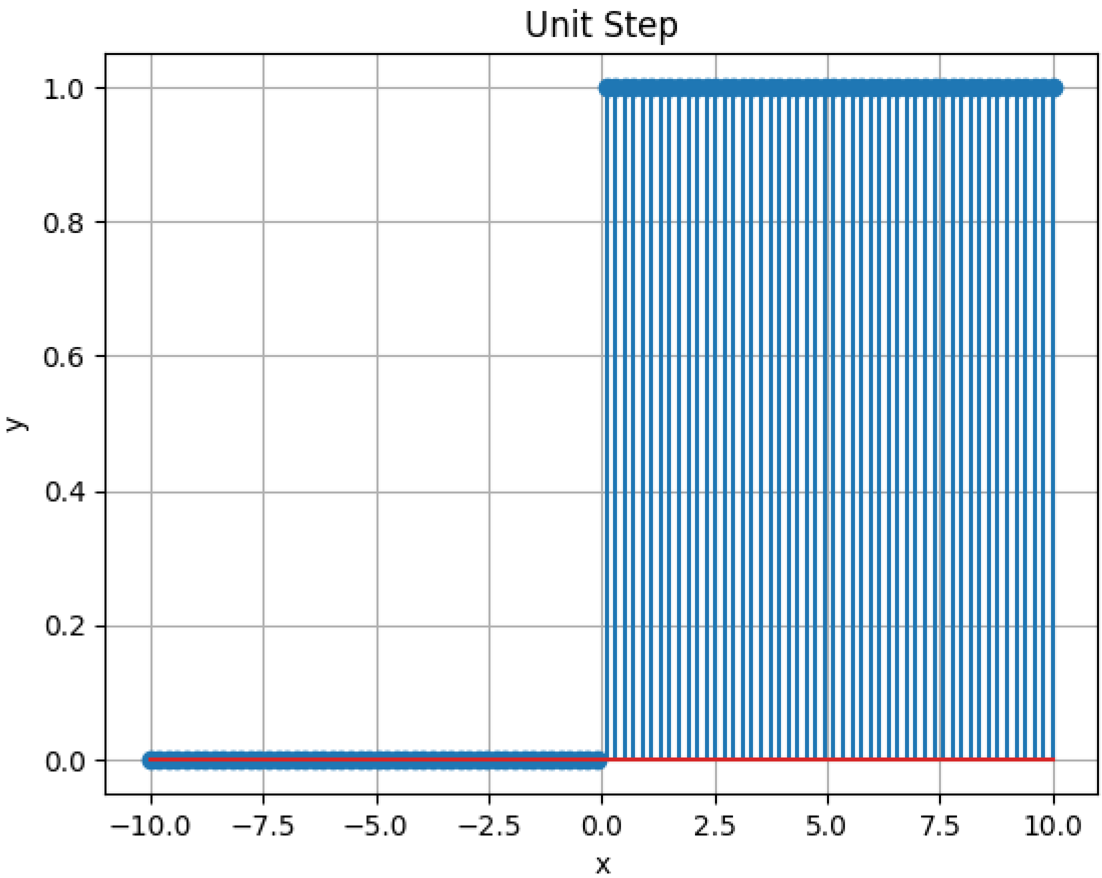
In [ ]:
```
#cosine wave
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0, 2 * np.pi, 1000)
y_cos = np.cos(x)

plt.plot(x, y_cos, label='cos(x)')

plt.title('Cosine Wave')
plt.xlabel('x')
plt.ylabel('cos(x)')
plt.grid(True)
plt.show()
```



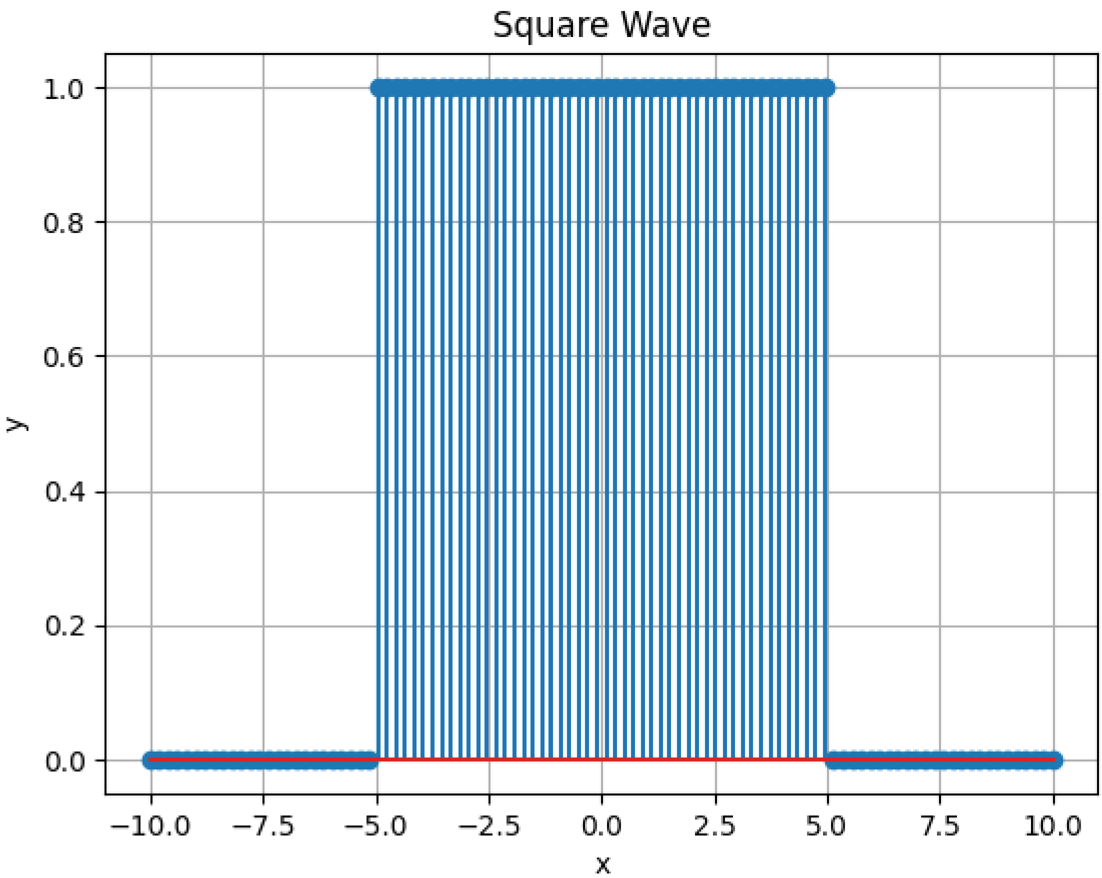

In [ ]:
```
#unit impulse
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-10, 10, 100)
y = np.zeros(len(x))
y[50] = 1

plt.stem(x, y)
plt.title('Unit Impulse')
plt.xlabel('x')
plt.ylabel('y')
plt.grid(True)
plt.show()
```



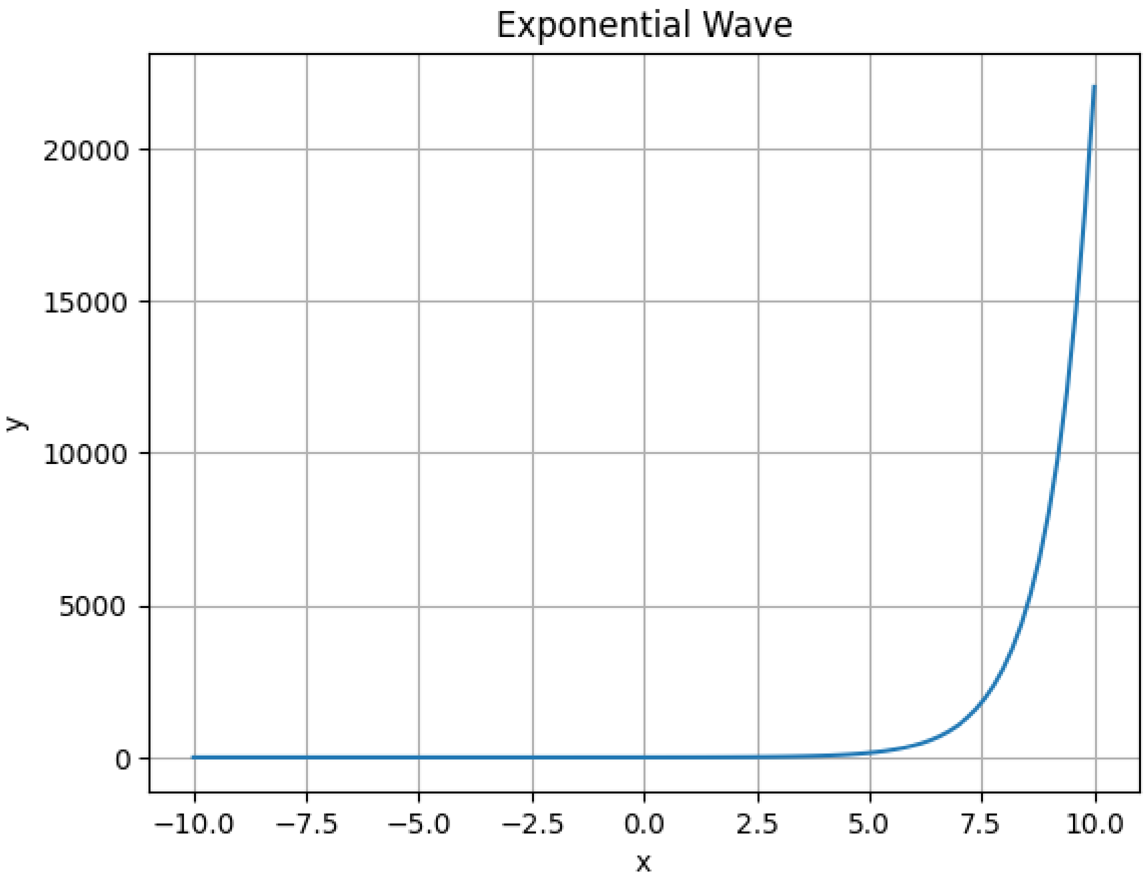

In [ ]:
```
#unit step wave
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-10, 10, 100)
y = np.zeros(len(x))
y[50:] = 1

plt.stem(x, y)
plt.title('Unit Step')
plt.xlabel('x')
plt.ylabel('y')
plt.grid(True)
plt.show()
```




In [ ]:
```
#square wave
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-10, 10, 100)
y = np.zeros(len(x))
y[25:75] = 1

plt.stem(x, y)
plt.title('Square Wave')
plt.xlabel('x')
plt.ylabel('y')
plt.grid(True)
plt.show()
```

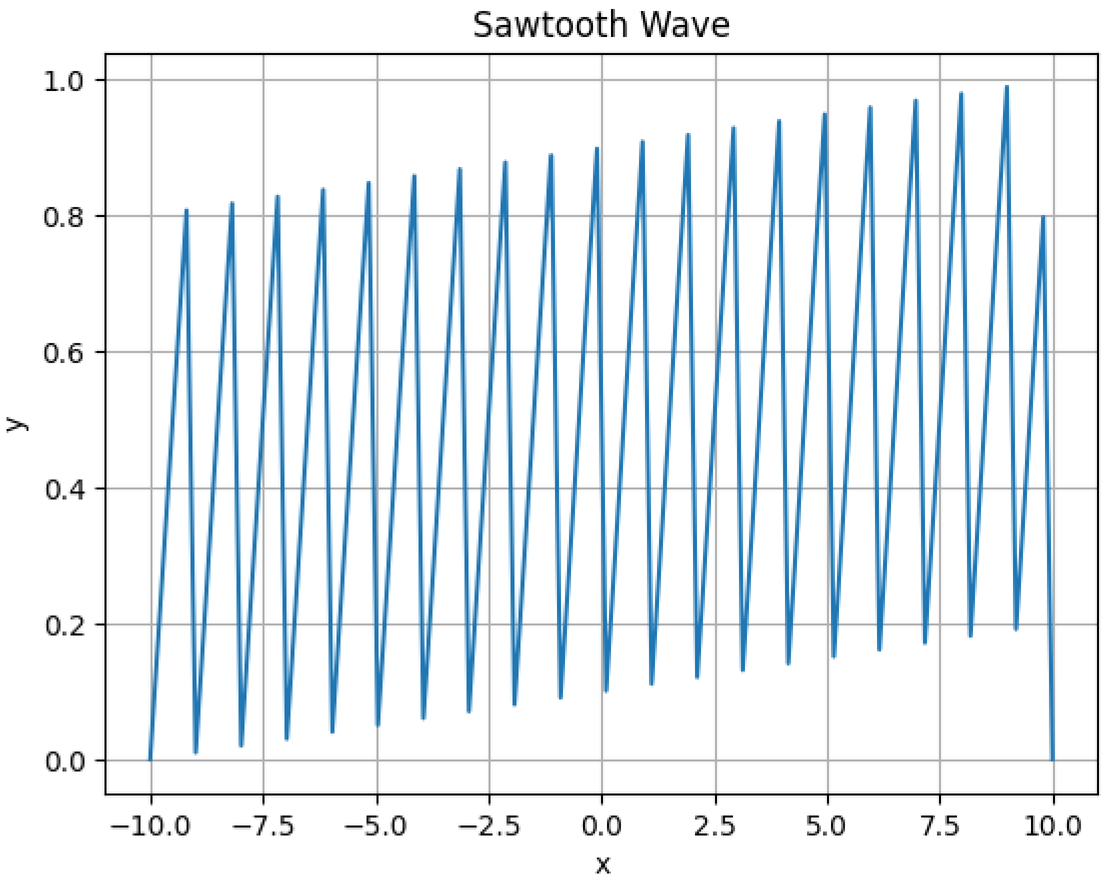



In [ ]:
```
#exponential wave
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-10, 10, 100)
y = np.exp(x)

plt.plot(x, y)
plt.title('Exponential Wave')
plt.xlabel('x')
plt.ylabel('y')
plt.grid(True)
plt.show()
```




In [ ]:

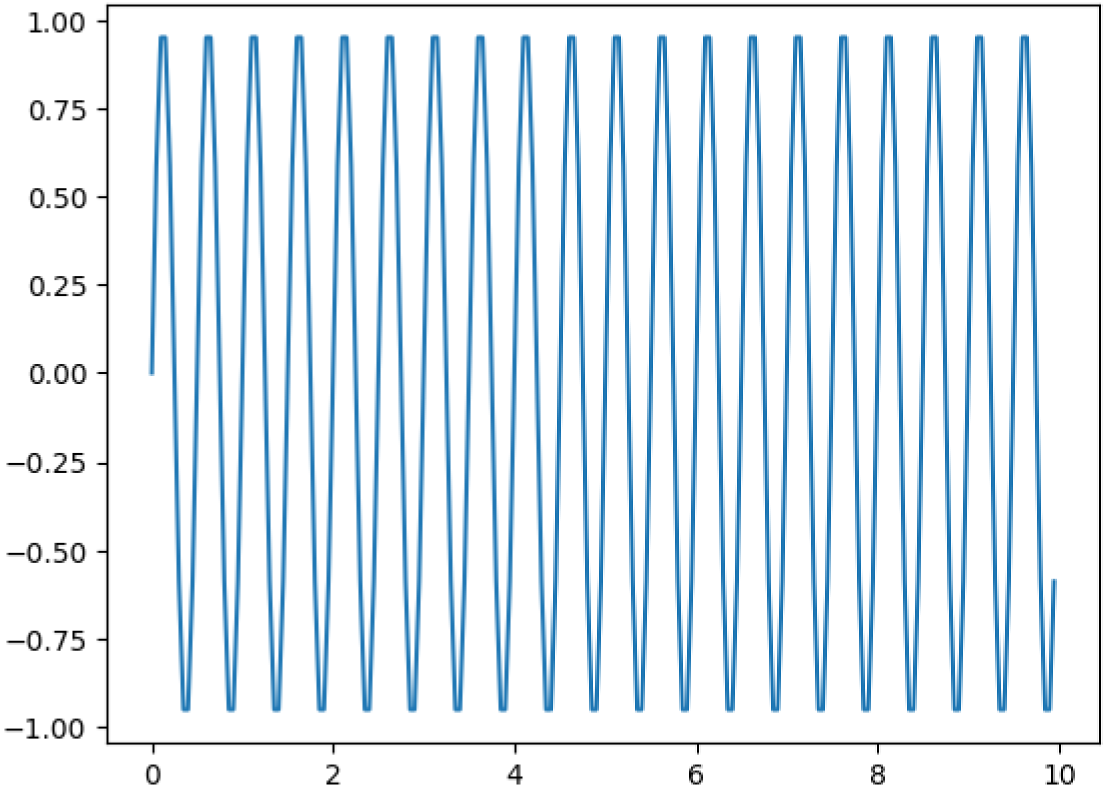
```
#sawtooth wave
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-10, 10, 100)
y = np.mod(x, 1)

plt.plot(x, y)
plt.title('Sawtooth Wave')
plt.xlabel('x')
plt.ylabel('y')
plt.grid(True)
plt.show()
```




**Fourier transformation**

3. Let's generate an audio signal. Explore the following code (Ex. Try varying the sample rate and see what happens).

In [ ]:
```
def generate_sine_wave(freq, sample_rate, duration):
  x = np.linspace(0,duration, sample_rate*duration, endpoint=False)
  frequencies = x*freq
  y = np.sin((2*np.pi)*frequencies)
  return x, y

sample_rate = 20
duration = 10
x,y = generate_sine_wave(2,sample_rate, duration)
plt.plot(x,y)
plt.show()
```




4. Now use generate_sine_wave() to generate two signals. How would you change pitch of the signal?

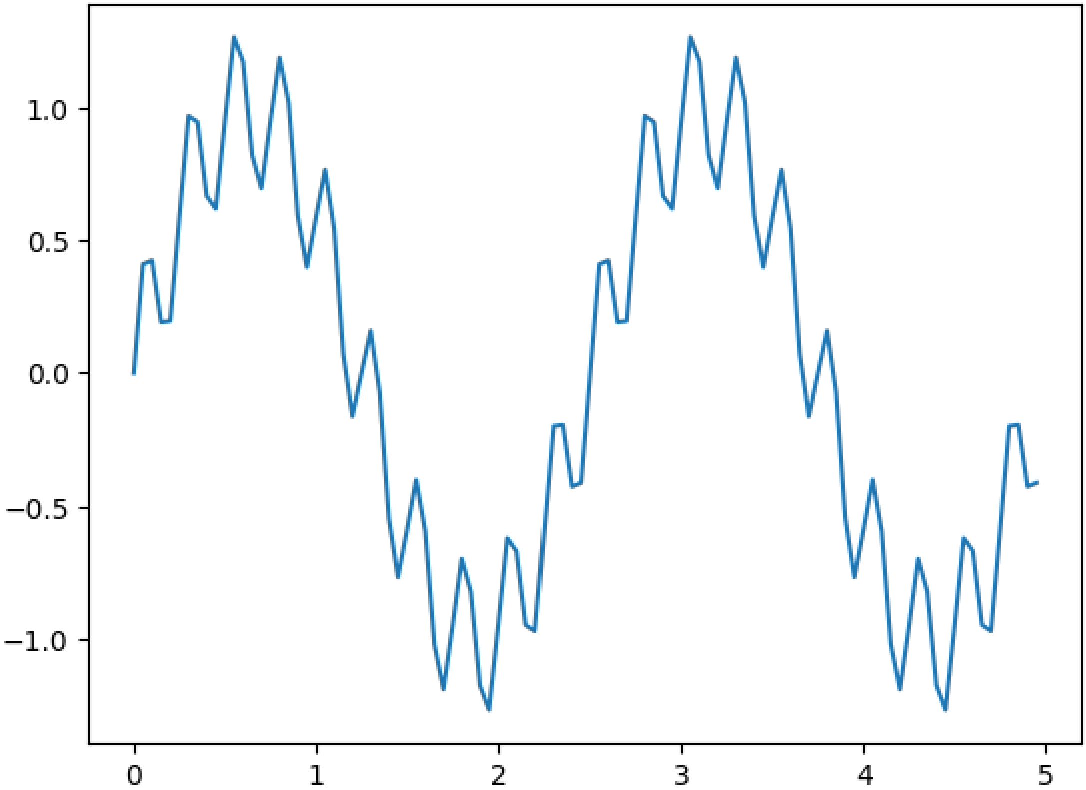
_, nice_tone = generate_sine_wave(#You may fill this)
_, noise_tone = generate_sine_wave(#You may fill this) #This will be an unwanted noise signal
noise_tone = noise_tone * 0.3 #Try changing the power of the noise signal

mixed_tone = nice_tone + noise_tone #Mix noise with the original signal

plt.plot(mixed_tone)
plt.show()

In [ ]:

```
nice_tone_x,nice_tone_y  = generate_sine_wave(0.4, 20, 5)
noise_tone_x, noise_tone_y = generate_sine_wave(4, 20, 5)

noise_tone_y = noise_tone_y * 0.3

mixed_tone = noise_tone_y + nice_tone_y

plt.plot(nice_tone_x,mixed_tone)
plt.show()
```




5. If you want, you can normalize and save it using the following code snippet [Optional].

from scipy.io.wavfile import write

#Normalization
normalized_tone = np.int16((mixed_tone / mixed_tone.max()) * 32767)
write("mysinewave.wav", SAMPLE_RATE, normalized_tone)

In [ ]:
```
from scipy.io.wavfile import write

SAMPLE_RATE = 20

normalized_tone = np.int16((mixed_tone / mixed_tone.max()) * 32767)

write("mysinewave.wav", SAMPLE_RATE, normalized_tone)
```

6. Now, let's implement the fourier transform. [Explore the functions fft() and fftfreq() in detail]

from scipy.fft import fft, fftfreq

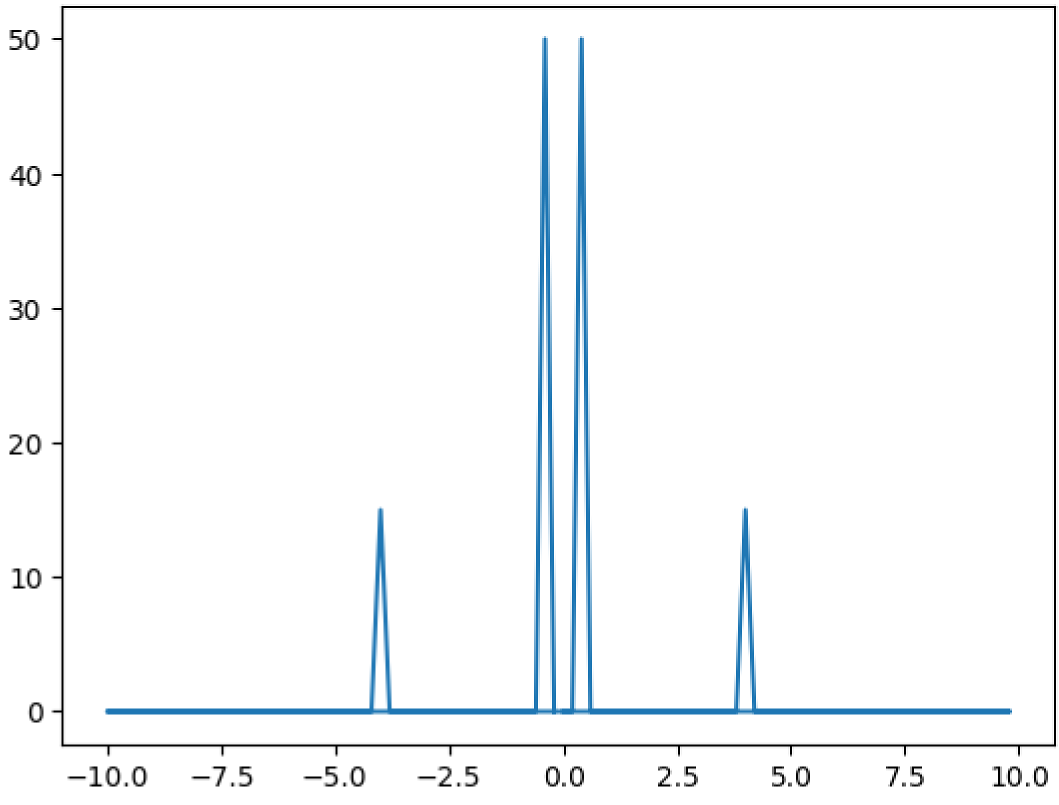
# Number of samples in normalized_tone
N = SAMPLE_RATE * DURATION

# Calculate the Fourier transform
yf = fft(mixed_tone)
xf = fftfreq(N, 1 / SAMPLE_RATE)

plt.plot(xf, np.abs(yf))
plt.show()

In [ ]:
```
from scipy.fft import fft, fftfreq

N = 20*5
```

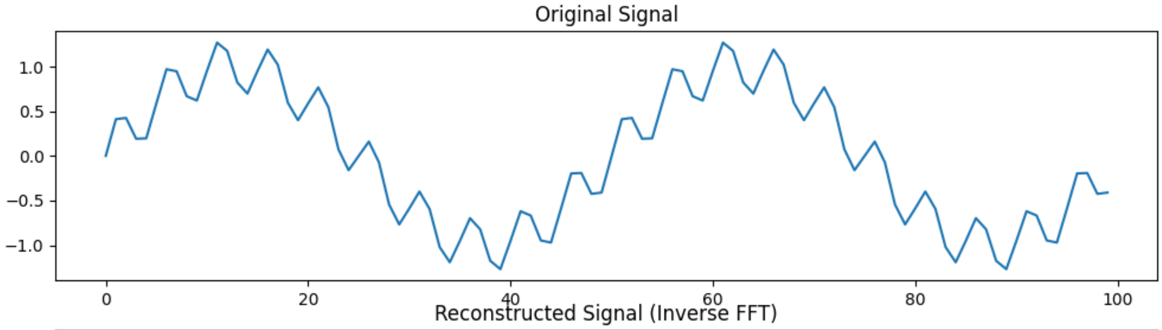
```
yf = fft(mixed_tone)
xf = fftfreq(N, 1 / sample_rate)

plt.plot(xf, np.abs(yf))
plt.show()
```

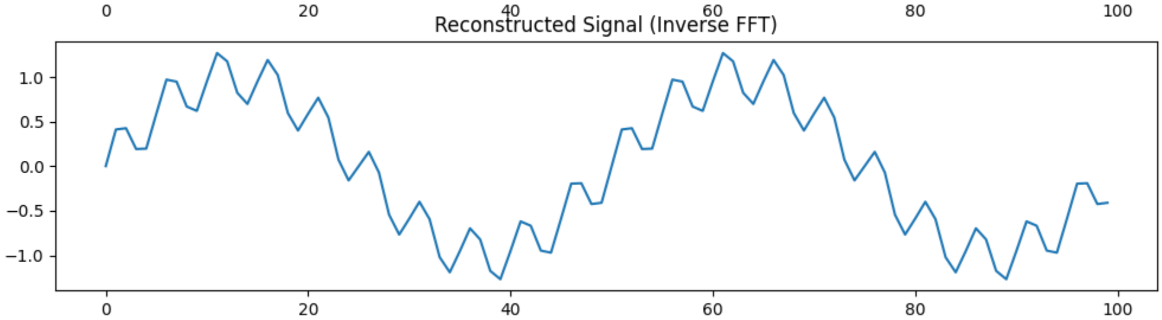



7. Use the function ifft() to reverse the operation. See how it reproduces the original signal.

In [ ]:
```
from scipy.fft import fft, fftfreq, ifft
import matplotlib.pyplot as plt

N = len(mixed_tone)
yf = fft(mixed_tone)
xf = fftfreq(N, 1 / sample_rate)

positive_frequencies = xf[:N//2]
magnitude_spectrum = np.abs(yf)[:N//2]


reconstructed_signal = ifft(yf)

plt.figure(figsize=(12, 6))

plt.subplot(2, 1, 1)
plt.plot(mixed_tone)
plt.title('Original Signal')

plt.subplot(2, 1, 2)
plt.plot(np.real(reconstructed_signal))
plt.title('Reconstructed Signal (Inverse FFT)')
```

```
plt.show()
```


Original Signal
1.0
0.5
0.0
−0.5
−1.0
0
20
40
60
80
100


Reconstructed Signal (Inverse FFT)
1.0
0.5
0.0
−0.5
−1.0
0
20
40
60
80
100