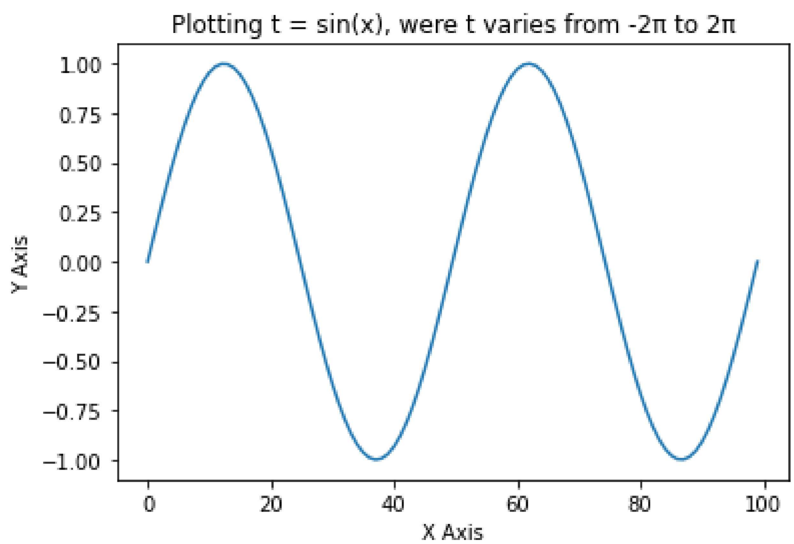
In [125]:
```
# Import Libraries
import numpy as np
import matplotlib.pyplot as plt
import math
import cmath
from sympy import *
```

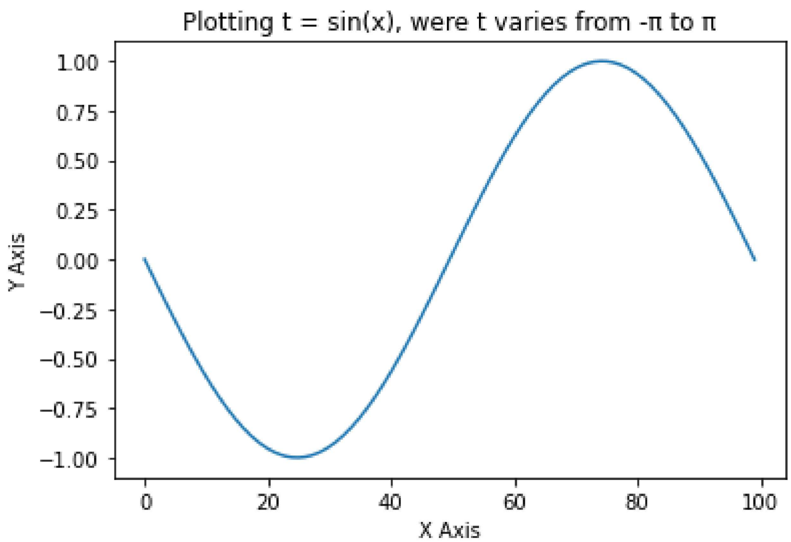
In [40]:
```
"""
Target for today:
1. Plotting a graph for t = sin(t), where t varies from -2π to 2π using matplotli
2. Generate a sinusoidal signal. Plot its ACF.
"""
```

Out[40]:
```
'\nTarget for today:\n1. Plotting a graph for t = sin(t), where t varies from -
2π to 2π using matplotlib and numpy\n2. Generate a sinusoidal signal. Plot its
ACF.\n'
```

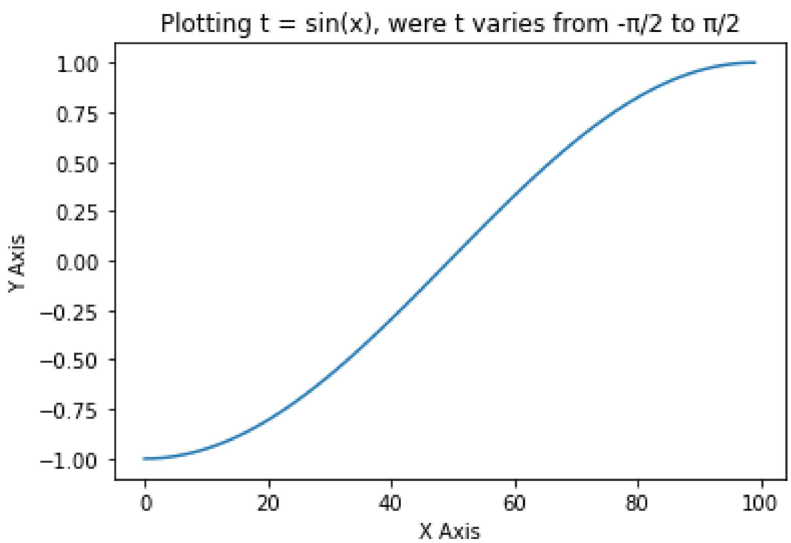
In [75]:
```
# 1. 1. Plotting a graph for t = sin(t), where t varies from -2π to 2π using matp

x = np.linspace(-2 * np.pi, 2 * np.pi, 100) # Plotting the graph for t = sin(t) v
# from -2π to 2π; 100 denotes Linearly spaced numbers
t = np.sin(x)

plt.title("Plotting t = sin(x), were t varies from -2π to 2π")
plt.xlabel("X Axis")
plt.ylabel("Y Axis")

plt.plot(t, marker = 'o')
plt.show()
```



In [79]:
```
x = np.linspace(-np.pi, np.pi, 100) # Plotting the graph for t = sin(t) where t v
# from -π to π; 100 denotes Linearly spaced numbers
t = np.sin(x)

plt.title("Plotting t = sin(x), were t varies from -π to π")
plt.xlabel("X Axis")
plt.ylabel("Y Axis")

plt.plot(t)
plt.show()
```



In [32]:
```
x = np.linspace(-np.pi/2, np.pi/2, 100) # Plotting the graph for t = sin(t) where
# from -π/2 to π/2; 100 denotes Linearly spaced numbers
t = np.sin(x)


plt.title("Plotting t = sin(x), were t varies from -π/2 to π/2")
plt.xlabel("X Axis")
plt.ylabel("Y Axis")
plt.plot(t)
plt.show()
```



In [50]:
```
# 2. Generate a sinusoidal signal. Plot its ACF.
def find_root(a, b, c):
    # The j in our solution stands for i (iota - complex numbers)
    # Calclulate Discriminant
    D = b**2 - (4*a*c)
    print("Discriminant Value = ", D)
    if D > 0:
        print("Roots are real and unique")
        x1 = (-b + cmath.sqrt(D))/(2 * a)
        x2 = (-b - cmath.sqrt(D))/(2 * a)
        print("The Roots are of the equation are :")
        print(x1)
        print(x2)
    elif D < 0:
        print("Roots are imaginary")
        x1 = (-b + cmath.sqrt(D))/(2 * a)
        x2 = (-b - cmath.sqrt(D))/(2 * a)
        print(x1)
        print(x2)
    elif D == 0:
        print("Roots are equal and real")
        x1 = (-b + cmath.sqrt(D))/(2 * a)
        x2 = (-b - cmath.sqrt(D))/(2 * a)
        print(x1)
        print(x2)
find_root(1, -5, 6)
```

```
Discriminant Value =  1
Roots are real and unique
The Roots are of the equation are :
(3+0j)
(2+0j)
```

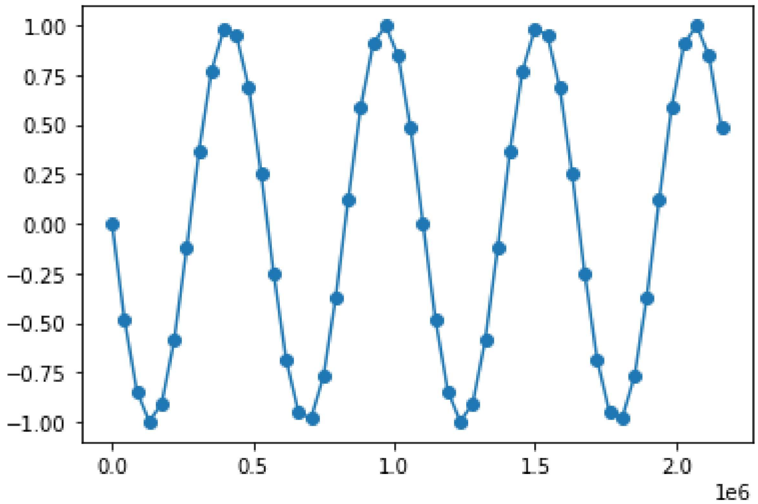
In [48]:
```
# Sir's version
def find_root(a, b, c):
    D = (b**2) - (4*a*c)
    x1 = (-b + cmath.sqrt(D))/(2 * a)
    x2 = (-b - cmath.sqrt(D))/(2 * a)
    return D, x1, x2
find_root(1, 1, 1)
```

Out[48]:
```
(-3, (-0.5+0.8660254037844386j), (-0.5-0.8660254037844386j))
```

In [66]:
```
# Function to generate a sine wave
def generate_sine_wave(freq, sample_rate, duration):
    x = np.linspace(freq, sample_rate * duration, endpoint = False)
    frequencies = x * freq
    # 2pi because np.sin takes input in radians
    y = np.sin((2 * np.pi) * frequencies)
    return x, y
```

In [87]:
```
# Generate  a 2 Hertz sine wave that lasts for 5 seconds
SAMPLE_RATE = 441000 # Hertz
DURATION = 5 # seconds
x, y = generate_sine_wave(2, SAMPLE_RATE, DURATION)
plt.plot(x, y, marker = 'o')
```

Out[87]:
```
[<matplotlib.lines.Line2D at 0x22e003276d0>]
```



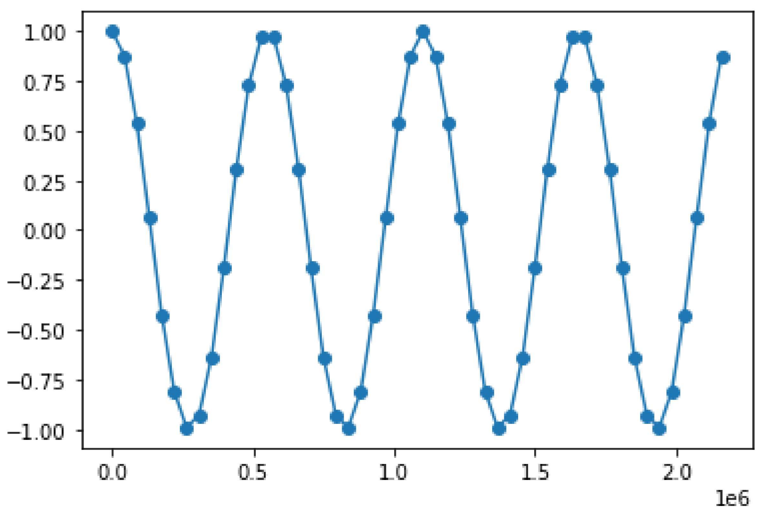
In [70]:
```
generate_sine_wave(1, 2, 3)
```

Out[70]:
```
(array([1. , 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2. , 2.1, 2.2,
        2.3, 2.4, 2.5, 2.6, 2.7, 2.8, 2.9, 3. , 3.1, 3.2, 3.3, 3.4, 3.5,
        3.6, 3.7, 3.8, 3.9, 4. , 4.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.7, 4.8,
        4.9, 5. , 5.1, 5.2, 5.3, 5.4, 5.5, 5.6, 5.7, 5.8, 5.9]),
 array([-2.44929360e-16,  5.87785252e-01,  9.51056516e-01,  9.51056516e-01,
         5.87785252e-01,  3.67394040e-16, -5.87785252e-01, -9.51056516e-01,
        -9.51056516e-01, -5.87785252e-01, -4.89858720e-16,  5.87785252e-01,
         9.51056516e-01,  9.51056516e-01,  5.87785252e-01,  6.12323400e-16,
        -5.87785252e-01, -9.51056516e-01, -9.51056516e-01, -5.87785252e-01,
        -7.34788079e-16,  5.87785252e-01,  9.51056516e-01,  9.51056516e-01,
         5.87785252e-01,  8.57252759e-16, -5.87785252e-01, -9.51056516e-01,
        -9.51056516e-01, -5.87785252e-01, -9.79717439e-16,  5.87785252e-01,
         9.51056516e-01,  9.51056516e-01,  5.87785252e-01,  1.10218212e-15,
        -5.87785252e-01, -9.51056516e-01, -9.51056516e-01, -5.87785252e-01,
        -1.22464680e-15,  5.87785252e-01,  9.51056516e-01,  9.51056516e-01,
         5.87785252e-01,  4.89982516e-15, -5.87785252e-01, -9.51056516e-01,
        -9.51056516e-01, -5.87785252e-01]))
```

In [71]:
```
# Function to generate a cosine wave
def generate_cosine_wave(freq, sample_rate, duration):
    x = np.linspace(freq, sample_rate * duration, endpoint = False)
    frequencies = x * freq
    # 2pi because np.sin takes input in radians
    y = np.cos((2 * np.pi) * frequencies)
    return x, y
```

In [81]:
```
# Generate  a 2 Hertz cosine wave that lasts for 5 seconds
SAMPLE_RATE = 441000 # Hertz
DURATION = 5 # seconds
x, y = generate_cosine_wave(2, SAMPLE_RATE, DURATION)
plt.plot(x, y, marker = 'o')
```

Out[81]: [<matplotlib.lines.Line2D at 0x22e017a54f0>]

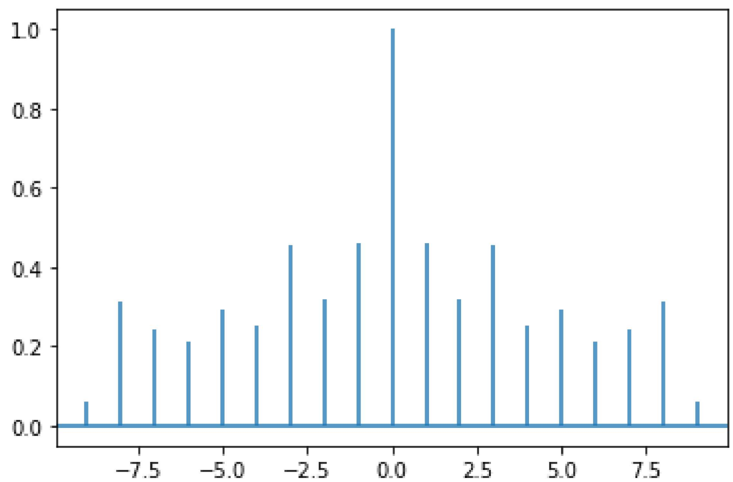


In [105]:
```
nice_tone = generate_sine_wave(400, SAMPLE_RATE, DURATION)
noise_tone = generate_sine_wave(4000, SAMPLE_RATE, DURATION)
mixed_tone = nice_tone + noise_tone

plt.show()
```

In [111]:
```
data = np.array([31.78, 55.65, 44.56])
```

In [138]:
```
data = np.array([24.40, 110.25, 20.05, 22.00, 61.90, 7.80, 15.00, 22.80, 34.90, 5

# Plot autocorrelation
plt.acorr(data, maxlags = 9)
```

Out[138]:
```
(array([-9, -8, -7, -6, -5, -4, -3, -2, -1,  0,  1,  2,  3,  4,  5,  6,  7,
         8,  9]),
 array([0.06138832, 0.31476972, 0.24381587, 0.21251556, 0.2904885 ,
        0.25378293, 0.45623263, 0.31893261, 0.45843318, 1.        ,
        0.45843318, 0.31893261, 0.45623263, 0.25378293, 0.2904885 ,
        0.21251556, 0.24381587, 0.31476972, 0.06138832]),
 <matplotlib.collections.LineCollection at 0x22e043c1100>,
 <matplotlib.lines.Line2D at 0x22e043c1820>)
```



In [127]:
```
x, y = symbols('x y')
```

In [129]:
```
integrate(x, x)
```

Out[129]: $\frac{x^2}{2}$

In [130]:
```
integrate(sin(x) + cos(x), x)
```

Out[130]: $\sin(x) - \cos(x)$

In [133]:
```
integrate(x ** 3)
```

Out[133]: $\frac{x^4}{4}$

In [134]:
```
integrate(exp(x))
```

Out[134]: e^x

In [135]:
```
integrate(x, (x, 0, 5)) # Limits of Integral of x from 0 to 5
```

Out[135]:

$$\frac{25}{2}$$

In [ ]: