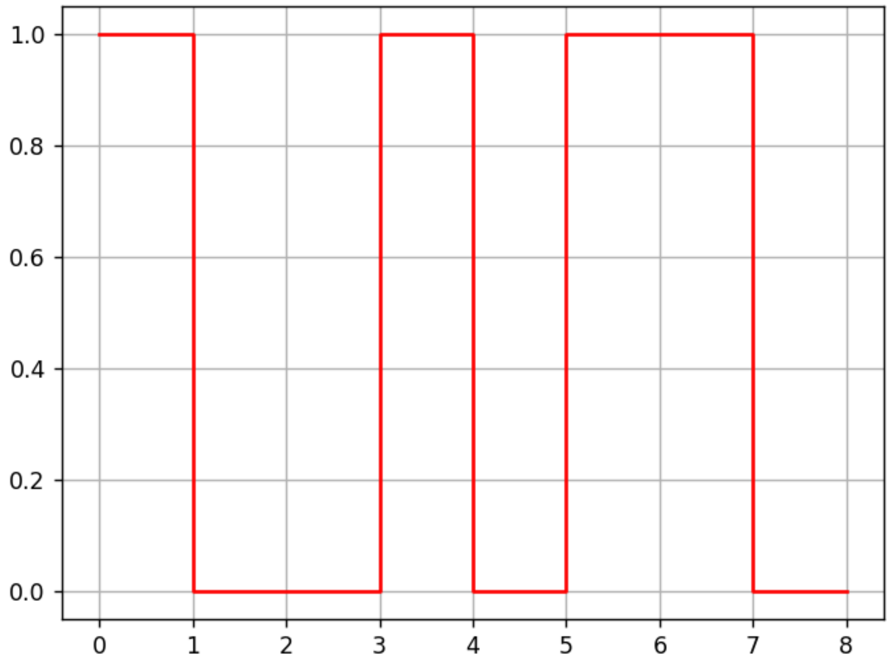
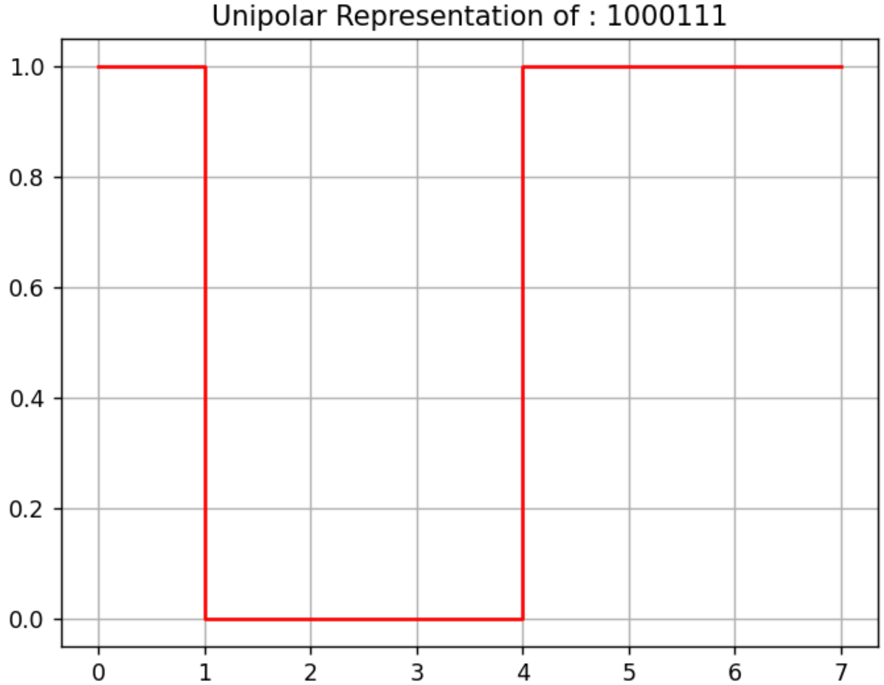
```python
import numpy as np
import matplotlib.pyplot as plt

inp_signal = input('Enter binary bit: ')

x = list()
y = list()
x.append(0)
y.append(1)

for i in inp_signal:
    if i == '0':
        if(y[-1] == 0):
            y.append(0)
        else:
            x.append(x[-1])
            y.append(0)
            y.append(0)
    else:
        if(y[-1] == 1):
            y.append(1)
        else:
            x.append(x[-1])
            y.append(1)
            y.append(1)

    x.append(x[-1]+1)

plt.title('Unipolar Representation of : '+inp_signal)
plt.plot(x,y,'red')
plt.grid(True)
plt.show()
```

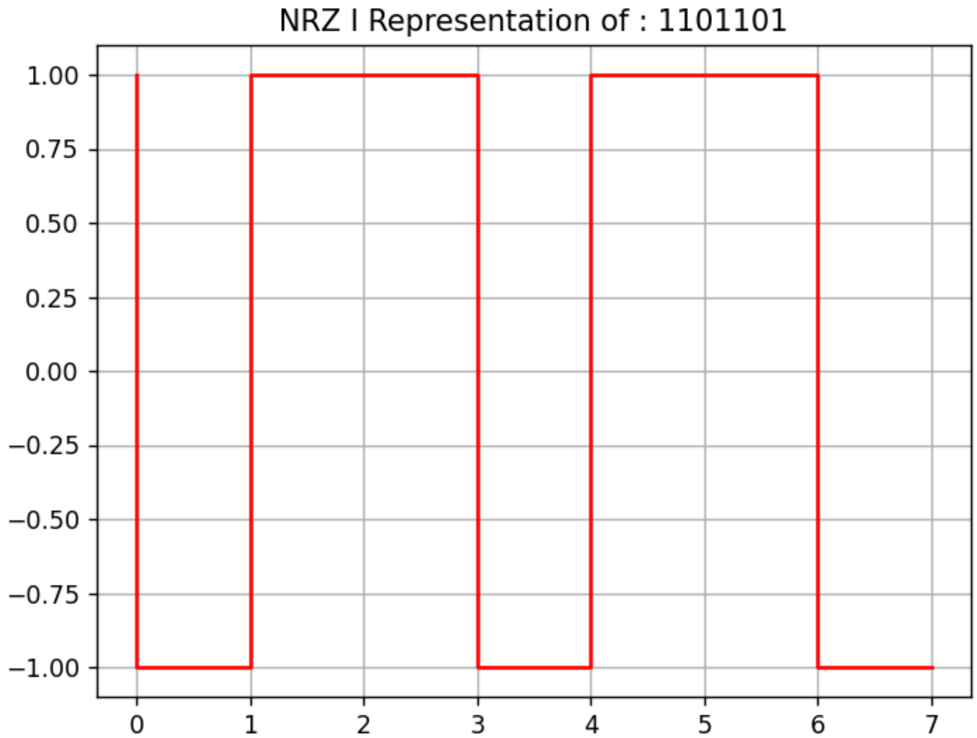
Unipolar Representation of : 10010110

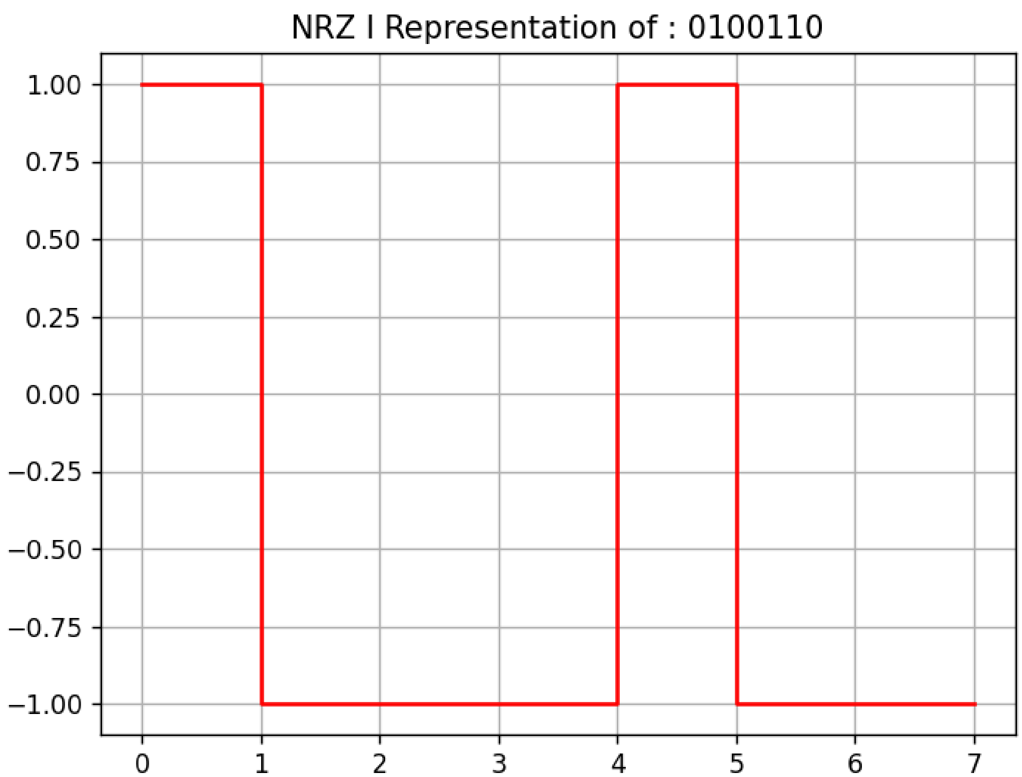


Unipolar Representation of : 1000111

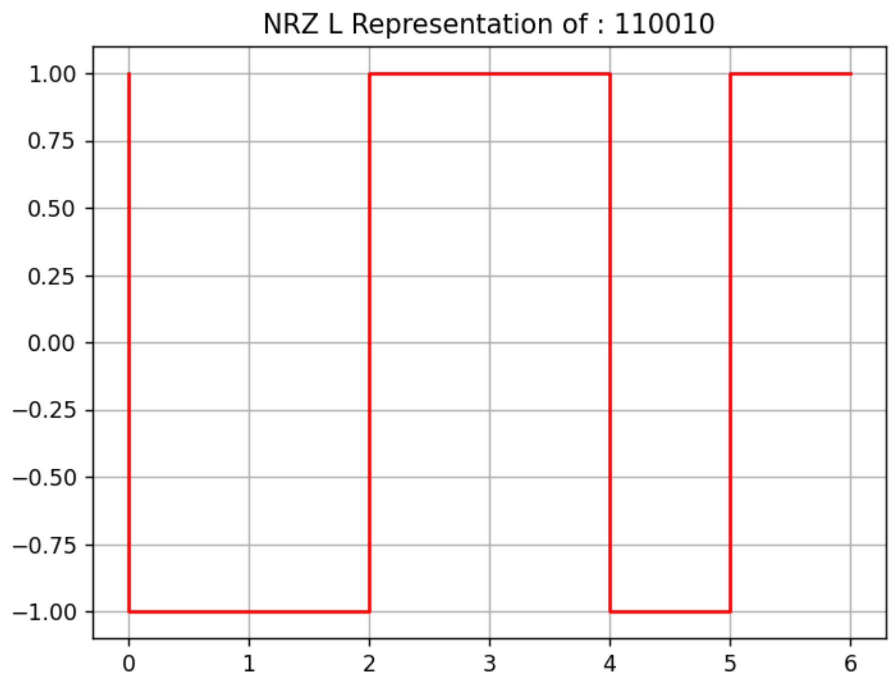


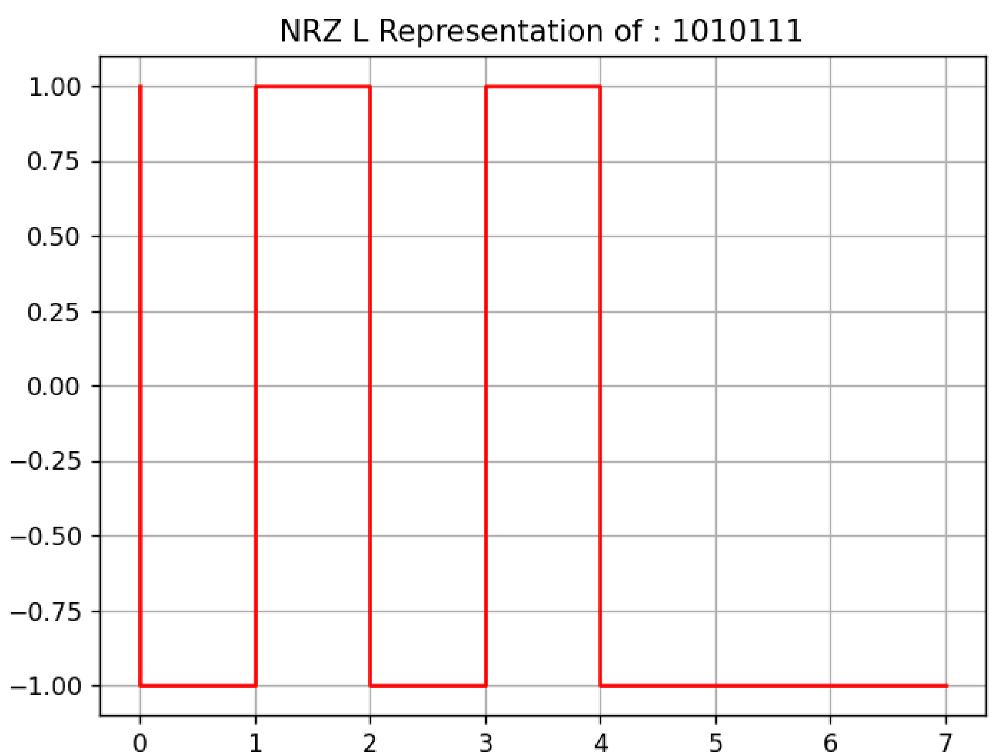
```python
import numpy as np
import matplotlib.pyplot as plt

inp_signal = input('Enter binary bit: ')

x = list()
y = list()
x.append(0)
y.append(1)

for i in inp_signal:
    if i == '0':
        y.append(y[-1])
    else:
        x.append(x[-1])
        y.append(-1*y[-1])
        y.append(y[-1])
    x.append(x[-1]+1)

plt.title('NRZ I Representation of : '+inp_signal)
plt.plot(x,y,'red')
plt.grid(True)
plt.show()
```


NRZ I Representation of : 1101101
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
1
2
3
4
5
6
7

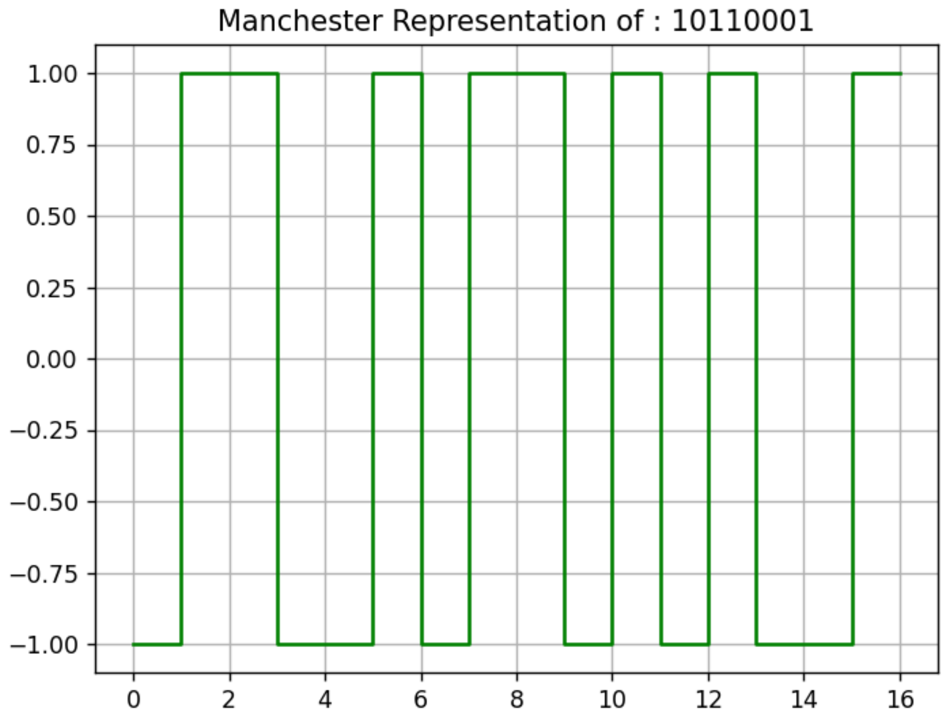

NRZ I Representation of : 0100110
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
1
2
3
4
5
6
7

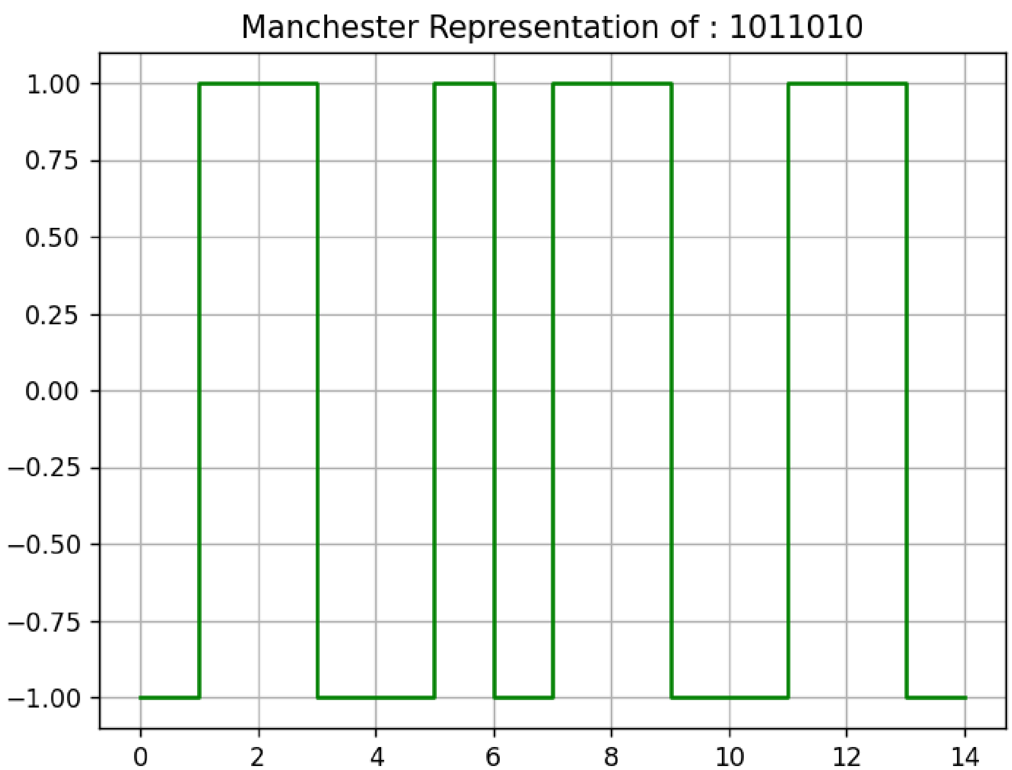
```python
import numpy as np
import matplotlib.pyplot as plt

inp_signal = input('Enter binary bit: ')

x = list()
y = list()
x.append(0)
y.append(1)

for i in inp_signal:
    if i == '0':
        if(y[-1] == 1):
            y.append(1)
        else:
            x.append(x[-1])
            y.append(1)
            y.append(1)
    else:
        if(y[-1] == 1):
            x.append(x[-1])
            y.append(-1)
            y.append(-1)
        else:
            y.append(-1)
    x.append(x[-1]+1)

plt.title('NRZ L Representation of : '+inp_signal)
plt.plot(x,y,'red')
plt.grid(True)
plt.show()
```


NRZ L Representation of : 110010
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
1
2
3
4
5
6

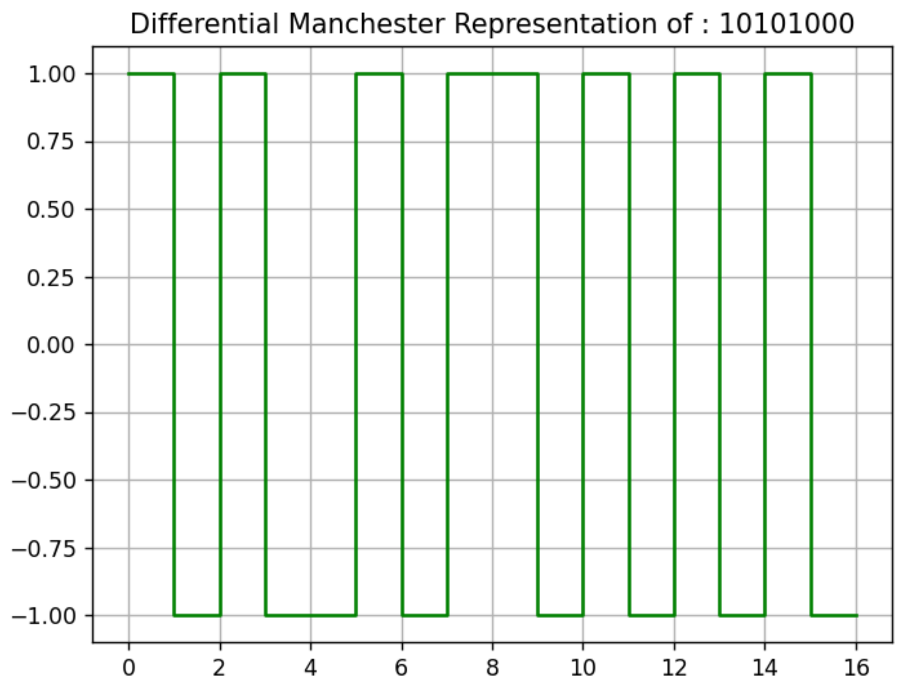

NRZ L Representation of : 1010111
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
1
2
3
4
5
6
7

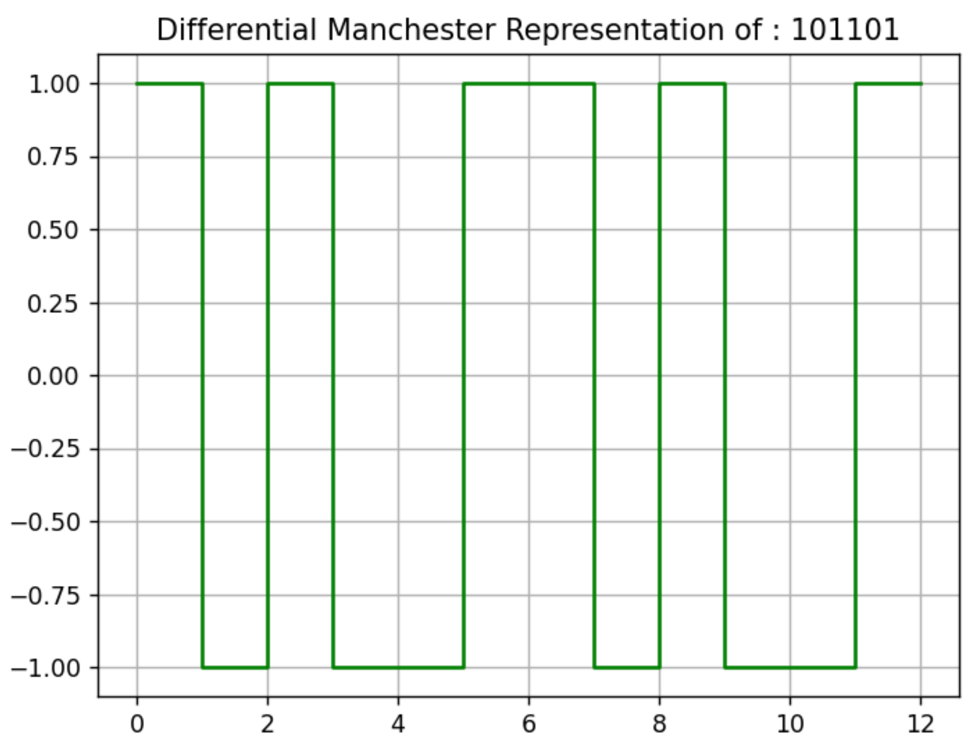
```python
import numpy as np
import matplotlib.pyplot as plt

inp_signal = input('Enter binary bit: ')

x = list()
y = list()

for i in inp_signal:
    if i == '0':
        y.append(1)
        y.append(-1)
    else:
        y.append(-1)
        y.append(1)

y.append(y[-1])
print('y : ',y)
plt.title('Manchester Representation of : '+inp_signal)
#plt.plot(x,y,'red')
x = np.arange(0,len(y))
plt.step(x,y,color='green',where='post')
plt.grid(True)
plt.show()
```


Manchester Representation of : 10110001
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
2
4
6
8
10
12
14
16

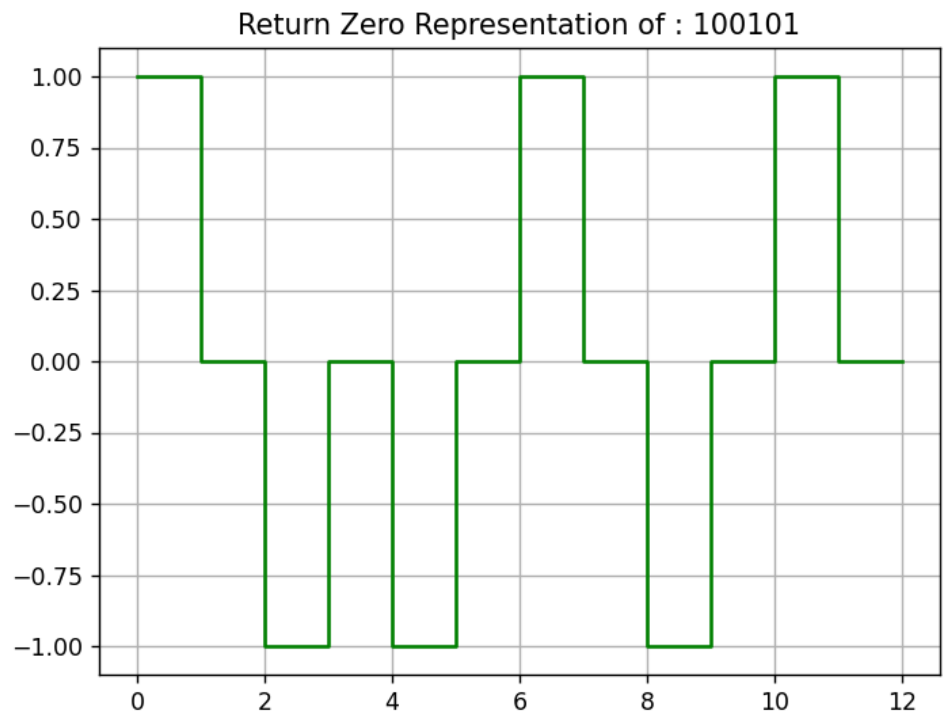

Manchester Representation of : 1011010
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
2
4
6
8
10
12
14

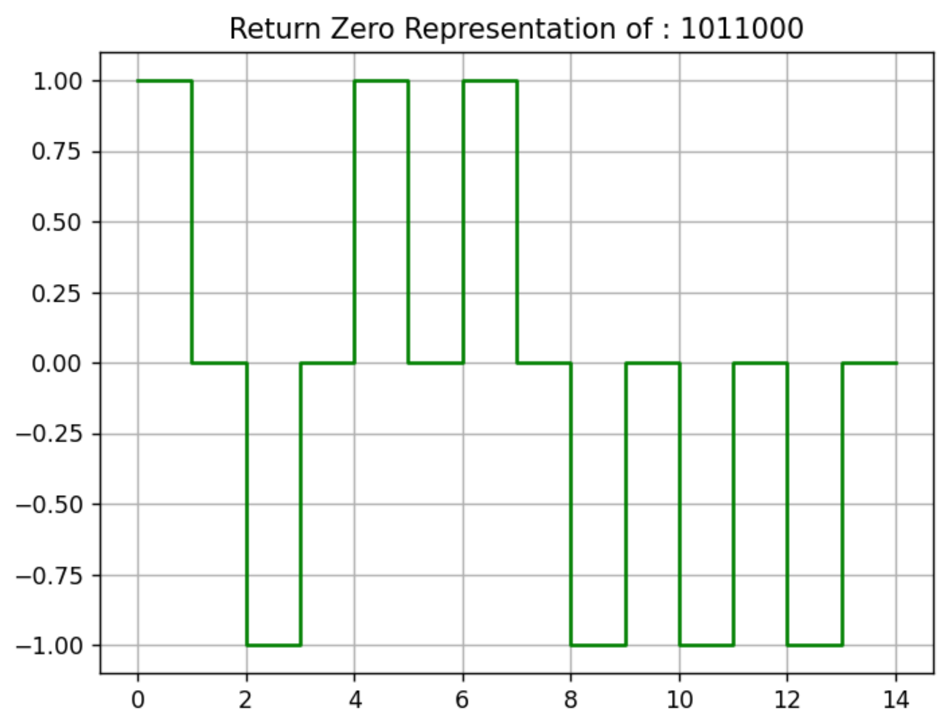
```python
import numpy as np
import matplotlib.pyplot as plt

inp_signal = input('Enter binary bit: ')

x = list()
y = list()
y.append(1)
for i in inp_signal:
    if i == '0':
        y.append(-1*y[-1])
        y.append(-1*y[-1])
    else:
        y.append(y[-1])
        y.append(-1*y[-1])

print('y : ',y)
plt.title('Differential Manchester Representation of : '+inp_signal)
#plt.plot(x,y,'red')
x = np.arange(0,len(y))
plt.step(x,y,color='green',where='pre')
plt.grid(True)
plt.show()
```


Differential Manchester Representation of : 10101000
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
2
4
6
8
10
12
14
16

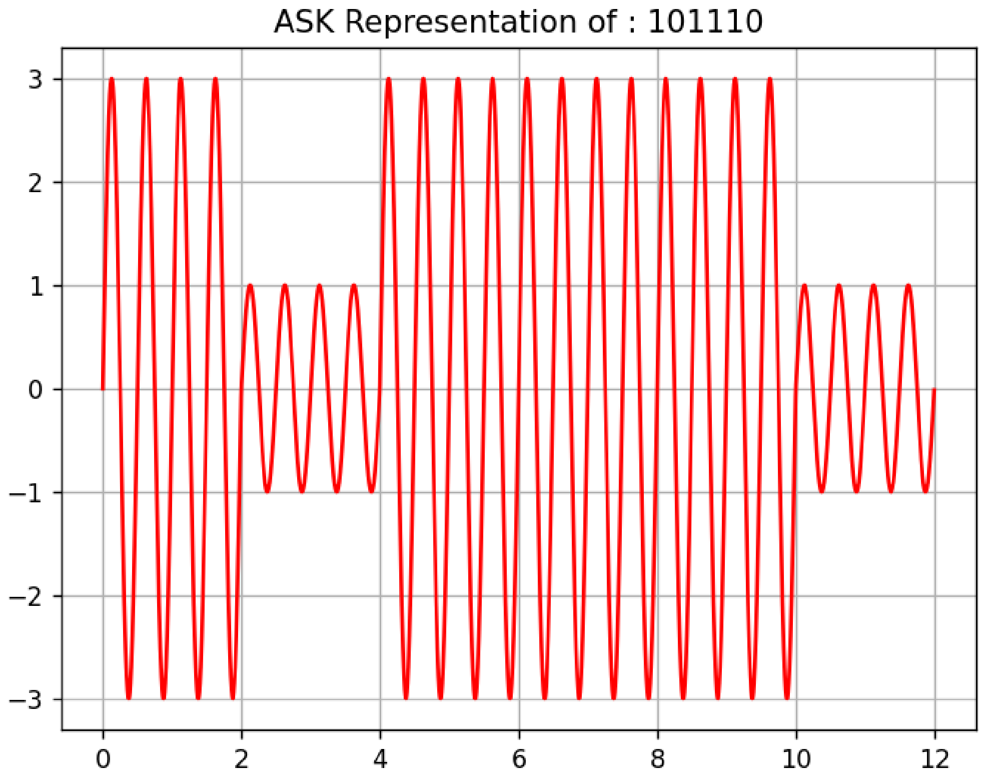

Differential Manchester Representation of : 101101
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
2
4
6
8
10
12

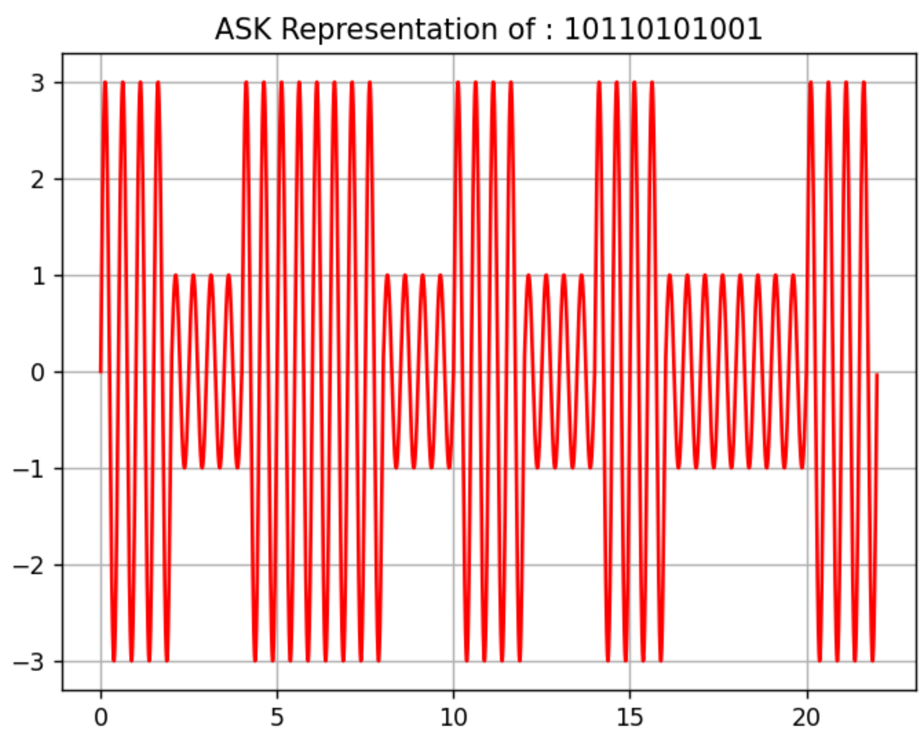
```python
import numpy as np
import matplotlib.pyplot as plt

inp_signal = input('Enter binary bit: ')

x = list()
y = list()
for i in inp_signal:
    if i == '0':
        y.append(-1)
        y.append(0)
    else:
        y.append(1)
        y.append(0)
y.append(y[-1])
print('y : ',y)
plt.title('Return Zero Representation of : '+inp_signal)
#plt.plot(x,y,'red')
x = np.arange(0,len(y))
plt.step(x,y,color='green',where='post')
plt.grid(True)
plt.show()
```


Return Zero Representation of : 100101
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
2
4
6
8
10
12

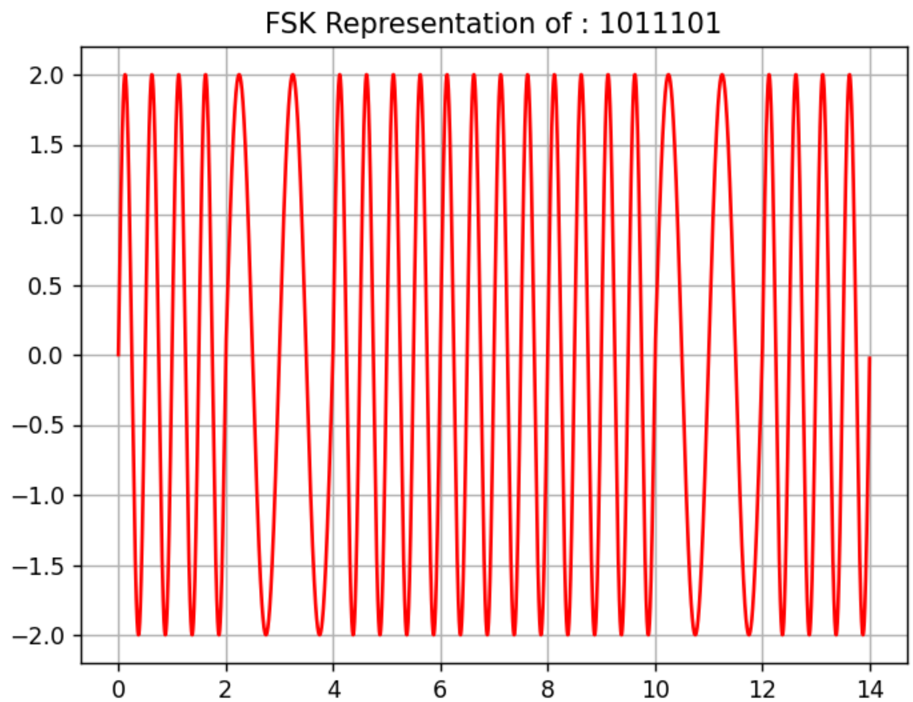

Return Zero Representation of : 1011000
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
2
4
6
8
10
12
14

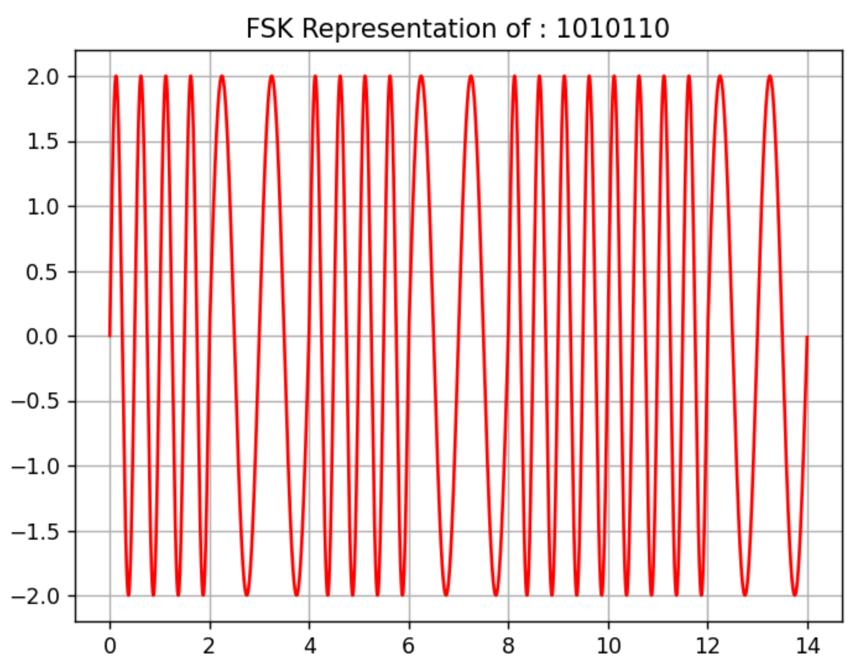
```python
import numpy as np
import matplotlib.pyplot as plt

inp_signal = input('Enter binary bit: ')

time = np.array(list())
amplitude = np.array(list())
step = 0
Low_amplitude = 1
High_amplitude = 3

def getSignal(A,time):
    frequency = 2
    phase = 0
    return A*np.sin(2*np.pi*frequency*time+phase)

for i in inp_signal:
    temp_time = np.arange(step,step+2,0.001)
    time = np.append(time,temp_time)
    if i == '0':
        amplitude = np.append(amplitude,getSignal(Low_amplitude,temp_time))
    else:
        amplitude = np.append(amplitude,getSignal(High_amplitude,temp_time))
    step = step + 2

plt.title('ASK Representation of : '+inp_signal)

plt.plot(time,amplitude,color='red')
plt.grid(True)
plt.show()
```


ASK Representation of : 101110
3
2
1
0
−1
−2
−3
0
2
4
6
8
10
12

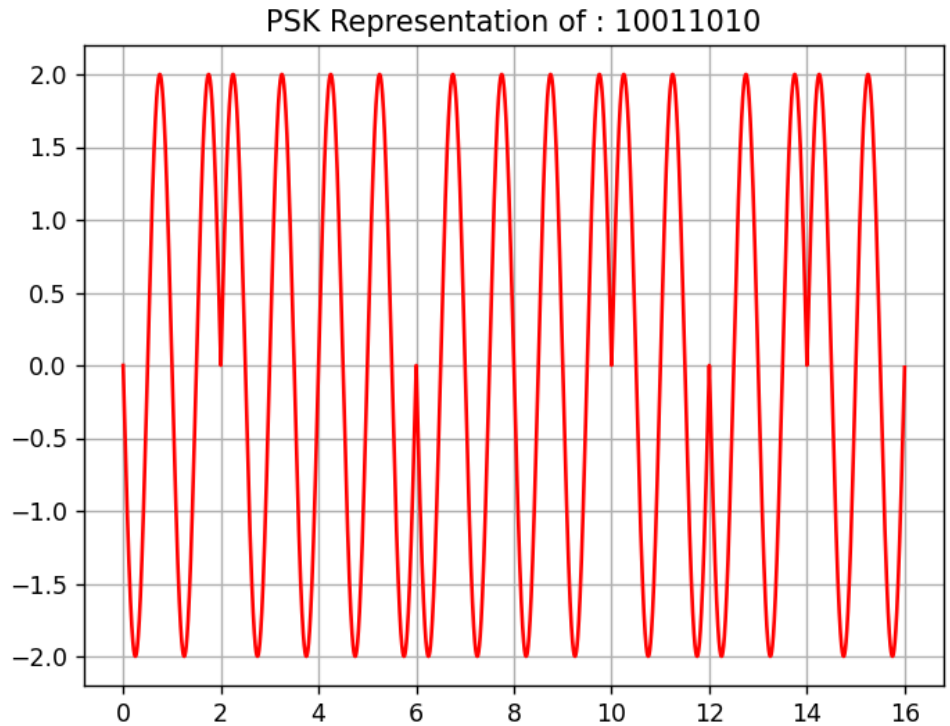

ASK Representation of : 10110101001
3
2
1
0
−1
−2
−3
0
5
10
15
20

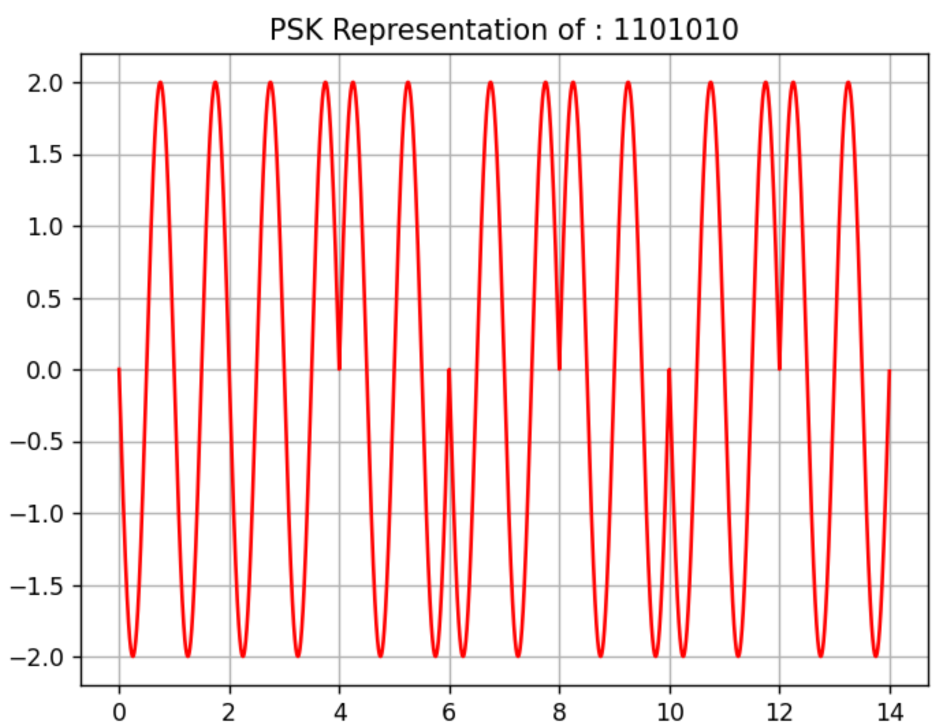
```python
import numpy as np
import matplotlib.pyplot as plt

inp_signal = input('Enter binary bit: ')

time = np.array(list())
amplitude = np.array(list())
step = 0
Low_frequency = 1
High_frequency = 2

def getSignal(f,time):
    A = 2
    phase = 0
    return A*np.sin(2*np.pi*f*time+phase)

for i in inp_signal:
    temp_time = np.arange(step,step+2,0.001)
    time = np.append(time,temp_time)
    if i == '0':
        amplitude = np.append(amplitude,getSignal(Low_frequency,temp_time))
    else:
        amplitude = np.append(amplitude,getSignal(High_frequency,temp_time))
    step = step + 2

plt.title('FSK Representation of : '+inp_signal)

plt.plot(time,amplitude,color='red')
plt.grid(True)
plt.show()
```


FSK Representation of : 1011101
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
0
2
4
6
8
10
12
14

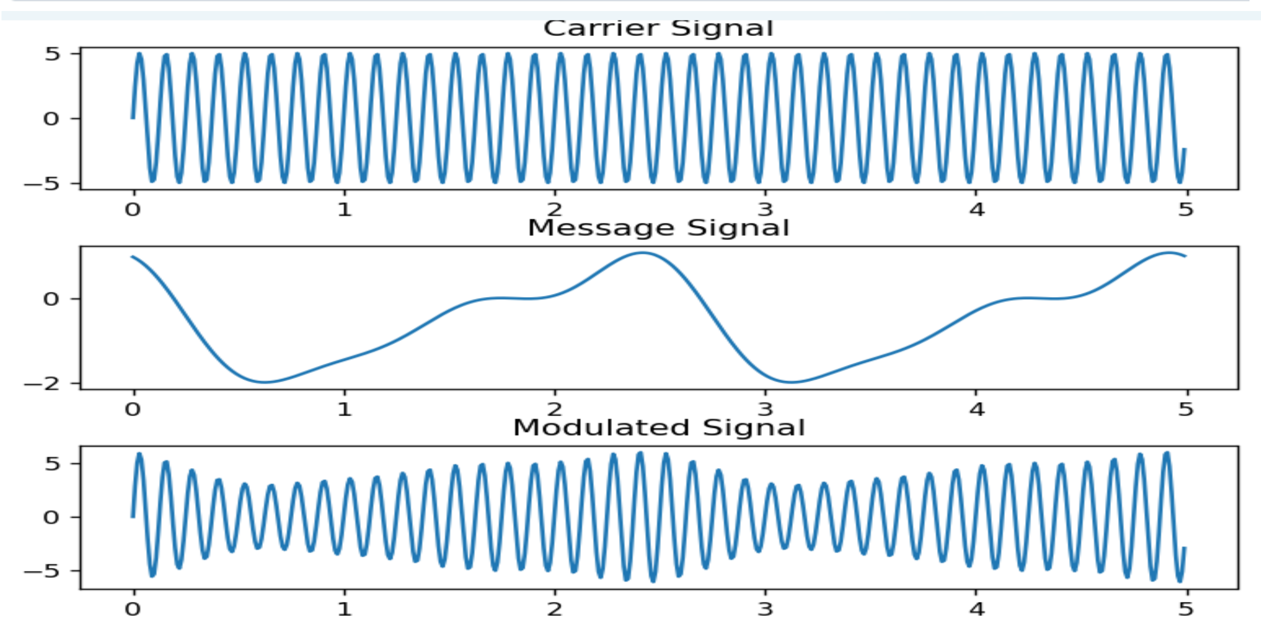

FSK Representation of : 1010110
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
0
2
4
6
8
10
12
14

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

inp_signal = input('Enter binary bit: ')

time = np.array(list())
amplitude = np.array(list())
step = 0


def getSignal(phase,time):
    A = 2
    f = 1
    return A*np.sin(2*np.pi*f*time+phase)

for i in inp_signal:
    temp_time = np.arange(step,step+2,0.001)
    time = np.append(time,temp_time)
    if i == '0':
        amplitude = np.append(amplitude,getSignal(0,temp_time))
    else:
        amplitude = np.append(amplitude,getSignal(np.pi,temp_time))
    step = step + 2

plt.title('PSK Representation of : '+inp_signal)
plt.plot(time,amplitude,color='red')
plt.grid(True)
plt.show()
```

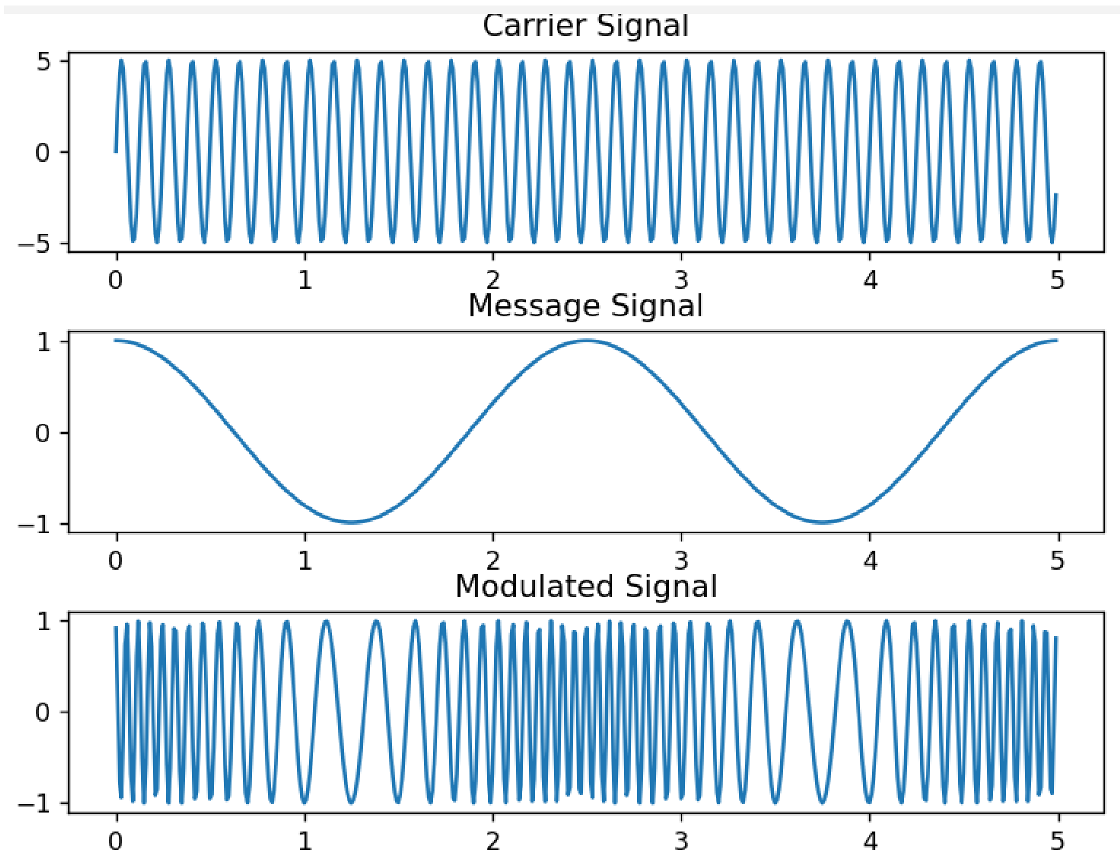

PSK Representation of : 10011010
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
0
2
4
6
8
10
12
14
16

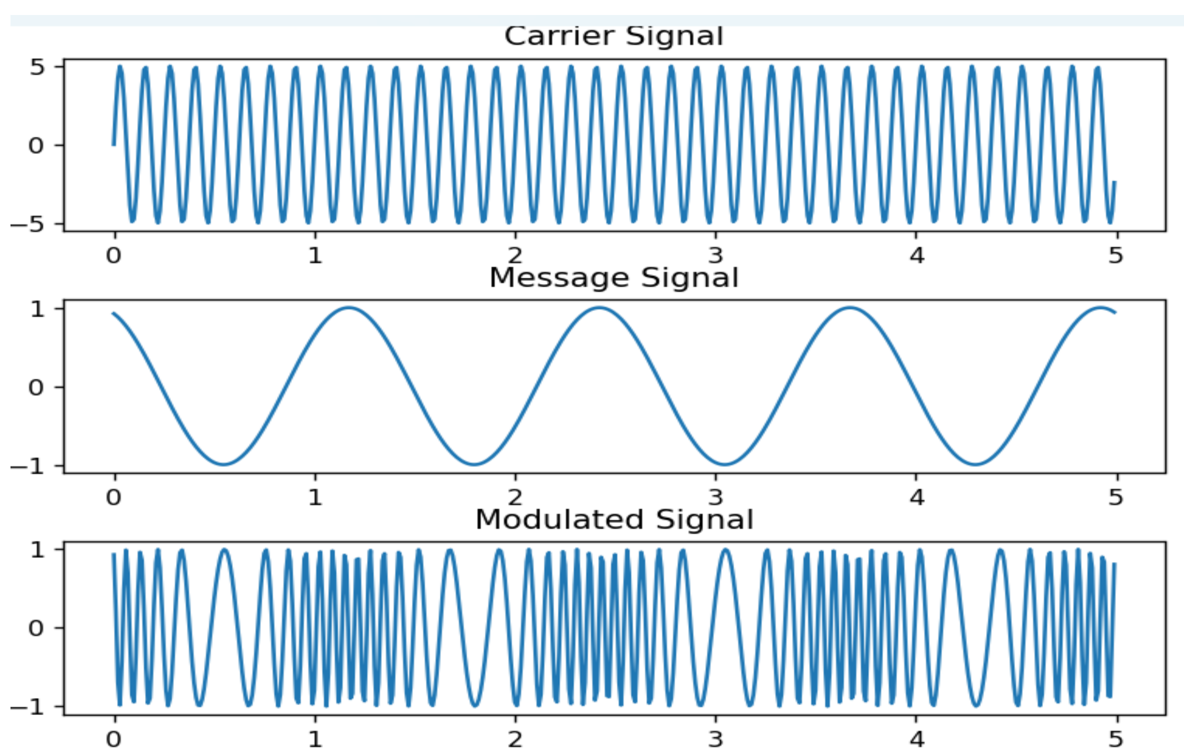

PSK Representation of : 1101010
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
0
2
4
6
8
10
12
14

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

figure, axis = plt.subplots(3)
plt.tight_layout()

#carrier Signal
carrier_time = np.arange(0,5,0.01)
carrier_A = 5
carrier_f = 8
carrier_phase = 0
carrier_amplitude = carrier_A*np.sin(2*np.pi*carrier_f*carrier_time+carrier_phase)
axis[0].plot(carrier_time,carrier_amplitude)
axis[0].set_title('Carrier Signal')

#Message Signal
message_time = np.arange(0,5,0.01)
message_A = 1
message_f = 0.4
message_phase = (np.pi/2)
thita = 2*np.pi*message_f*message_time+message_phase
message_amplitude = message_A*(np.sin(thita)**3-np.cos(thita)**2+np.cos(thita))
axis[1].plot(message_time,message_amplitude)
axis[1].set_title('Message Signal')

#Modulated Signal
modulated_time = np.arange(0,5,0.01)
modulated_amplitude = (carrier_A + message_amplitude)*np.sin(2*np.pi*carrier_f*carrier_time+carrier_phase)
axis[2].plot(modulated_time,modulated_amplitude)
axis[2].set_title('Modulated Signal')

plt.show()
```



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

figure, axis = plt.subplots(3)
plt.tight_layout()

#carrier Signal
carrier_time = np.arange(0,5,0.01)
carrier_A = 5
carrier_f = 8
carrier_phase = 0
carrier_amplitude = carrier_A*np.sin(2*np.pi*carrier_f*carrier_time+carrier_phase)
axis[0].plot(carrier_time,carrier_amplitude)
axis[0].set_title('Carrier Signal')

#Message Signal
message_time = np.arange(0,5,0.01)
message_A = 1
message_f = 0.4
message_phase = (np.pi/2)
thita = 2*np.pi*message_f*message_time+message_phase
message_amplitude = message_A*np.sin(thita)
axis[1].plot(message_time,message_amplitude)
axis[1].set_title('Message Signal')

#Modulated Signal
modulated_time = np.arange(0,5,0.01)
moudlated_frequency = 10
k = 0.4 #sensitivity
phi = 2*np.pi*moudlated_frequency*modulated_time + k*np.cumsum(message_amplitude)
modulated_amplitude = np.cos(phi)
axis[2].plot(modulated_time,modulated_amplitude)
axis[2].set_title('Modulated Signal')

plt.show()
```


Carrier Signal
Message Signal
Modulated Signal


Carrier Signal
Message Signal
Modulated Signal

```python
import numpy as np
inp_signal = input('Enter Message Signal : ')
m = len(inp_signal)
r = int(np.log2(m))
while(2**r < m+r+1):
    r = r + 1
l = m+r
print('-> Message len :',m,' -> Redundant len :',r,' -> Total len :',l)
hammingCode = list()
sig_ind = 0
for i in range(1,l+1):
    if (np.log2(i)-int(np.log2(i)) == 0.0):
        hammingCode.append(0)
    else:
        hammingCode.append(ord(inp_signal[sig_ind])-48)
        sig_ind = sig_ind + 1
print('Before Calculating Redundant Bit : ',hammingCode)
for i in range(1,l+1):
    if (np.log2(i)-int(np.log2(i)) == 0.0):
        print('\t',i,'th bit parity calculation')
        parity = 0
        ind = i - 1
        while ind < l:
            for j in range(i):
                if ind >= l:
                    break
                parity = parity + hammingCode[ind]
                print('\t\t',ind+1,'th bit',hammingCode[ind],'total parity :',parity)
                ind = ind + 1
            for j in range(i):
                if ind >= l:
                    break
                ind = ind + 1
        hammingCode[i-1] = parity%2
        print('\t',i,'th Parity will :',hammingCode[i-1])
print('-----------Generated Hamming code is------------')
print(hammingCode)
```

```
+ Enter Message Signal : 1011101
+ -> Message len : 7  -> Redundant len : 4  -> Total len : 11
+ Before Calculating Redundant Bit :  [0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1]
+         1 th bit parity calculation
+                 1 th bit 0 total parity : 0
+                 3 th bit 1 total parity : 1
+                 5 th bit 0 total parity : 1
+                 7 th bit 1 total parity : 2
+                 9 th bit 1 total parity : 3
+                 11 th bit 1 total parity : 4
+         1 th Parity will : 0
+         2 th bit parity calculation
+                 2 th bit 0 total parity : 0
+                 3 th bit 1 total parity : 1
+                 6 th bit 1 total parity : 2
+                 7 th bit 1 total parity : 3
+                 10 th bit 0 total parity : 3
+                 11 th bit 1 total parity : 4
+         2 th Parity will : 0
+         4 th bit parity calculation
+                 4 th bit 0 total parity : 0
+                 5 th bit 0 total parity : 0
+                 6 th bit 1 total parity : 1
+                 7 th bit 1 total parity : 2
+         4 th Parity will : 0
+         8 th bit parity calculation
+                 8 th bit 0 total parity : 0
+                 9 th bit 1 total parity : 1
+                 10 th bit 0 total parity : 1
+                 11 th bit 1 total parity : 2
+         8 th Parity will : 0
+ -----------Generated Hamming code is------------
+ [0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1]
```

```python
import numpy as np

inp_signal = input('Enter Hamming Code:')

hammingCode = list()
for i in inp_signal:
    hammingCode.append(ord(i)-48)

print('Hamming code is :',hammingCode)
l = len(inp_signal)
parity_mismatch = list()
for i in range(1,l+1):
    if (np.log2(i)-int(np.log2(i))) == 0.0:
        print(i,'th bit parity calculation')
        parity = 0
        ind = i - 1
        while ind < l:
            for j in range(i):
                if ind >= l:
                    break
                parity = parity + hammingCode[ind]
                print('\t',ind+1,'th bit',hammingCode[ind],'total parity :',parity)
                ind = ind + 1
            for j in range(i):
                if ind >= l:
                    break
                ind = ind + 1
        parity_mismatch.append(parity%2)
        print(i,'th Parity will :',parity_mismatch[-1])

print('----------Hamming Missmatch Parity code is------------')
print(parity_mismatch)
```

```
Enter Message Signal : 1011001
-> Message len : 7  -> Redundant len : 4  -> Total len : 11
Before Calculating Redundant Bit :  [0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1]
        1 th bit parity calculation
               1 th bit 0 total parity : 0
               3 th bit 1 total parity : 1
               5 th bit 0 total parity : 1
               7 th bit 1 total parity : 2
               9 th bit 0 total parity : 2
               11 th bit 1 total parity : 3
        1 th Parity will : 1
        2 th bit parity calculation
               2 th bit 0 total parity : 0
               3 th bit 1 total parity : 1
               6 th bit 1 total parity : 2
               7 th bit 1 total parity : 3
               10 th bit 0 total parity : 3
               11 th bit 1 total parity : 4
        2 th Parity will : 0
        4 th bit parity calculation
               4 th bit 0 total parity : 0
               5 th bit 0 total parity : 0
        7 th bit 1 total parity : 2
4 th Parity will : 0
8 th bit parity calculation
        8 th bit 0 total parity : 0
        9 th bit 0 total parity : 0
        10 th bit 0 total parity : 0
        11 th bit 1 total parity : 1
8 th Parity will : 1
-----------Hamming Missmatch Parity code is------------
[0, 0, 0, 1]
```