

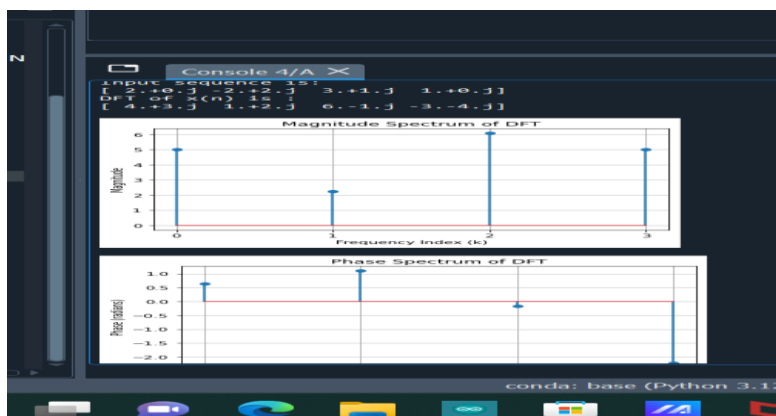
#DFT OF GIVEN SEQUENCE EXP-1

```
import numpy as np
import matplotlib.pyplot as plt
x = np.array([2, -2+2j, 3+1j, 1])
N = len(x)
X = np.zeros(N, dtype=complex)

for k in range(N):
    for n in range(N):
        X[k] += x[n] * np.exp(-2j * np.pi * k * n / N)
print("Input sequence is:")
print(x)
print("DFT of x(n) is :")
print(X)
```

```
magnitude_spectrum = np.abs(X)
# Plot the magnitude spectrum
plt.stem(np.arange(N), magnitude_spectrum)
plt.title('Magnitude Spectrum of DFT')
plt.xlabel('Frequency Index (k)')
plt.ylabel('Magnitude')
plt.xticks(np.arange(N))
plt.grid(True)
plt.show()
```

```
phase_spectrum = np.angle(X)
# Plot the phase spectrum
plt.stem(np.arange(N), phase_spectrum)
plt.title('Phase Spectrum of DFT')
plt.xlabel('Frequency Index (k)')
plt.ylabel('Phase (radians)')
plt.xticks(np.arange(N))
plt.grid(True)
plt.show()
```



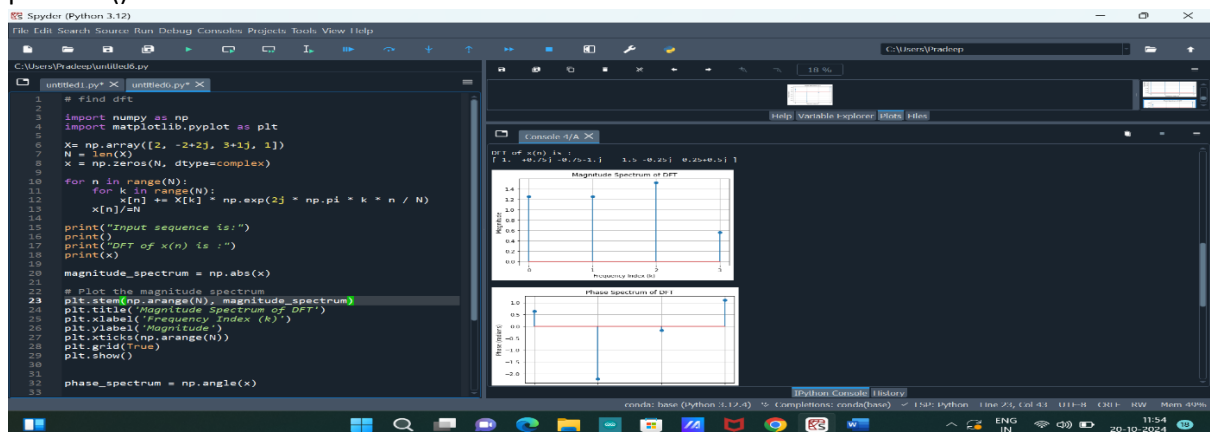
IDFT OF GIVEN SEQUENCE EXP-1

```
import numpy as np
import matplotlib.pyplot as plt
X= np.array([2, -2+2j, 3+1j, 1])
N = len(X)
x = np.zeros(N, dtype=complex)

for n in range(N):
    for k in range(N):
        x[n] += X[k] * np.exp(2j * np.pi * k * n / N)
    x[n]/=N
print("Input sequence is:")
print()
print("DFT of x(n) is :")
print(x)
```

```
magnitude_spectrum = np.abs(x)
# Plot the magnitude spectrum
plt.stem(np.arange(N), magnitude_spectrum)
plt.title('Magnitude Spectrum of DFT')
plt.xlabel('Frequency Index (k)')
plt.ylabel('Magnitude')
plt.xticks(np.arange(N))
plt.grid(True)
plt.show()
```

```
phase_spectrum = np.angle(x)
# Plot the phase spectrum
plt.stem(np.arange(N), phase_spectrum)
plt.title('Phase Spectrum of DFT')
plt.xlabel('Frequency Index (k)')
plt.ylabel('Phase (radians)')
plt.xticks(np.arange(N))
plt.grid(True)
plt.show()
```



Perform circular convolution EXP-2

```
import numpy as np
import matplotlib.pyplot as plt
def dft_matrix(N):
    # Construct the DFT matrix
    W = np.exp(-2j * np.pi / N)
    return np.array([[W**(i * j) for j in range(N)] for i in range(N)])
def idft_matrix(N):
    # Construct the IDFT matrix
    W = np.exp(2j * np.pi / N)
    return np.array([[W**(i * j) for j in range(N)] for i in range(N)]) / N
def dft(x):
    N = len(x)
    DFT = dft_matrix(N)
    return np.dot(DFT, x), DFT
def idft(X):
    N = len(X)
    IDFT = idft_matrix(N)
    return np.dot(IDFT, X), IDFT
# Input sequences
x1 = np.array([1, 2, 1, 2])
x2 = np.array([4, 3, 2, 1])
# Compute the DFT of the sequences
X1, DFT1 = dft(x1)
X2, DFT2 = dft(x2)
# Multiply the DFT outputs element-wise
X_product = X1 * X2
# Compute the IDFT of the product
x_product, IDFT = idft(X_product)
# Print the results
print("Twiddle factor matrix for DFT (N=4):")
print(np.round(DFT1.real, decimals=2) + 1j * np.round(DFT1.imag, decimals=2))
print("\nTwiddle factor matrix for IDFT (N=4):")
print(np.round(IDFT.real, decimals=2) + 1j * np.round(IDFT.imag, decimals=2))
print("\nX1(k) =", np.round(X1, decimals=2))
print("X2(k) =", np.round(X2, decimals=2))
print("Y(k) =", np.round(X_product, decimals=2))
print("y(n) =", np.round(x_product, decimals=2))
# Plot the magnitude and phase of the IDFT result
magnitude = np.abs(x_product)
phase = np.angle(x_product)
plt.figure(figsize=(12, 6))
# Plot magnitude
plt.subplot(2, 1, 1)
plt.stem(magnitude) # Removed use_line_collection=True
plt.title('Magnitude of IDFT')
plt.xlabel('Sample')
```

```
plt.ylabel('Magnitude')
# Plot phase
plt.subplot(2, 1, 2)
plt.stem(phase) # Removed use_line_collection=True
plt.title('Phase of IDFT')
plt.xlabel('Sample')
plt.ylabel('Phase (radians)')
plt.tight_layout()
plt.show()
```

The screenshot shows the Spyder IDE interface. The left pane contains the Python script, and the right pane shows the console output. The script defines functions for DFT and IDFT, computes the DFT of a sequence, and then computes the IDFT. It also plots the magnitude and phase of the IDFT result.

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 def dft_matrix(N):
4     # Construct the DFT matrix
5     W = np.exp(-2j * np.pi / N)
6     return np.array([[W**(i * j) for j in range(N)] for i in range(N)])
7 def idft_matrix(N):
8     # Construct the IDFT matrix
9     W = np.exp(2j * np.pi / N)
10    return np.array([[W**(i * j) for j in range(N)] for i in range(N)]) / N
11 def dft(x):
12     N = len(x)
13     DFT = dft_matrix(N)
14     return np.dot(DFT, x), DFT
15 def idft(X):
16     N = len(X)
17     IDFT = idft_matrix(N)
18     return np.dot(IDFT, X), IDFT
19 # Input sequences
20 x1 = np.array([1, 2, 1, 2])
21 x2 = np.array([4, 3, 2, 1])
22 # Compute the DFT of the sequences
23 X1, DFT1 = dft(x1)
24 X2, DFT2 = dft(x2)
25 # Multiply the DFT outputs element-wise
26 X_product = X1 * X2
27 # Compute the IDFT of the product
28 x_product, IDFT = idft(X_product)
29 # Print the results
30 print("Twiddle factor matrix for DFT (N=4):")
31 print(np.round(DFT1.real, decimals=2) + 1j * np.round(DFT1.imag, decimals=2))
32 print("\nTwiddle factor matrix for IDFT (N=4):")
33 print(np.round(IDFT.real, decimals=2) + 1j * np.round(IDFT.imag, decimals=2))
34 print("\nx1(k) =", np.round(X1, decimals=2))
35 print("\nx2(k) =", np.round(X2, decimals=2))
36 print("\ny(k) =", np.round(X_product, decimals=2))
37 print("\ny(n) =", np.round(x_product, decimals=2))
38 # Plot the magnitude and phase of the IDFT result
39 magnitude = np.abs(x_product)
40 phase = np.angle(x_product)
41 plt.figure(figsize=(12, 6))
42 # Plot magnitude
43 plt.subplot(2, 1, 1)
44 plt.stem(magnitude) # Removed use_line_collection=True
45 plt.title('Magnitude of IDFT')
46 plt.xlabel('Sample')
47 plt.ylabel('Magnitude')
48 # Plot phase
49 plt.subplot(2, 1, 2)
50 plt.stem(phase) # Removed use_line_collection=True
51 plt.title('Phase of IDFT')
52 plt.xlabel('Sample')
53 plt.ylabel('Phase (radians)')
54 plt.tight_layout()
55 plt.show()
```

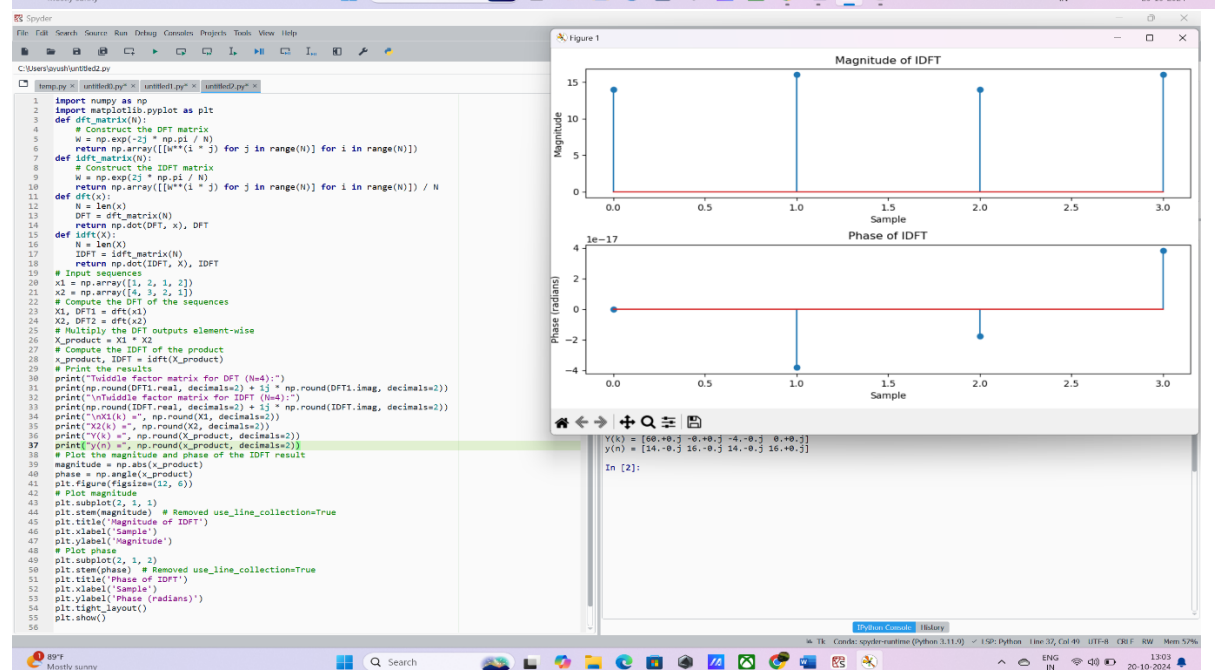
Console Output:

```
Python 3.11.0 | packaged by conda-forge | (main, Apr 19 2024, 18:27:10) [MSC v.1938 64 bit (AMD64)]
Type "copyright", "credits" or "license()" for more information.

Python 8.27.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: %runfile C:/Users/ayush/untitled2.py --dir
Twiddle factor matrix for DFT (N=4):
[[ 1.+0.j  1.+0.j  1.+0.j  1.+0.j]
 [ 1.+0.j  0.-1.j  1.+0.j  0.+1.j]
 [ 1.+0.j -1.+0.j  1.+0.j -1.+0.j]
 [ 1.+0.j  0.+1.j -1.+0.j  0.-1.j]]
Twiddle factor matrix for IDFT (N=4):
[[ 0.25+0.j  0.25+0.j  0.25+0.j  0.25+0.j]
 [ 0.25+0.j  0. +0.25j -0.25+0.j  0. +0.25j]
 [ 0.25+0.j -0.25+0.j  0.25+0.j -0.25+0.j]
 [ 0.25+0.j -0. -0.25j -0.25+0.j  0. +0.25j]]
x1(k) = [ 6.+0.j -0.-0.j -2.-0.j  0.-0.j]
x2(k) = [10.+0.j  2.-2.j  2.-0.j  2.+2.j]
y(k) = [60.+0.j -0.+0.j -4.-0.j  0.+0.j]
y(n) = [14.-0.j 16.-0.j 14.-0.j 16.+0.j]

In [2]:
```



DFT Using DIT FFT EXP-3

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

# Taking the sequence
print("Enter the sequence:")
x0 = complex(input("x(0)="))
x1 = complex(input("x(1)="))
x2 = complex(input("x(2)="))
x3 = complex(input("x(3)="))
x4 = complex(input("x(4)="))
x5 = complex(input("x(5)="))
x6 = complex(input("x(6)="))
x7 = complex(input("x(7)="))

# Stage 1
a = x0 + x4
b = x0 - x4
c = x2 + x6
d = x2 - x6
e = x1 + x5
f = x1 - x5
g = x3 + x7
h = x3 - x7

# Stage 2
i = a + np.exp(-2j * np.pi * 0 / 8) * c
j = b + np.exp(-2j * np.pi * 2 / 8) * d
k = a - np.exp(-2j * np.pi * 0 / 8) * c
l = b - np.exp(-2j * np.pi * 2 / 8) * d
m = e + np.exp(-2j * np.pi * 0 / 8) * g
n = f + np.exp(-2j * np.pi * 2 / 8) * h
o = e - np.exp(-2j * np.pi * 0 / 8) * g
p = f - np.exp(-2j * np.pi * 2 / 8) * h

# Stage 3
X0 = i + np.exp(-2j * np.pi * 0 / 8) * m
X1 = j + np.exp(-2j * np.pi * 1 / 8) * n
X2 = k + np.exp(-2j * np.pi * 2 / 8) * o
X3 = l + np.exp(-2j * np.pi * 3 / 8) * p
X4 = i - np.exp(-2j * np.pi * 0 / 8) * m
X5 = j - np.exp(-2j * np.pi * 1 / 8) * n
X6 = k - np.exp(-2j * np.pi * 2 / 8) * o
X7 = l - np.exp(-2j * np.pi * 3 / 8) * p

# Store in a NumPy array
X = np.array([X0, X1, X2, X3, X4, X5, X6, X7])
```

```
# Print the array
print("X[K] =", X)
```

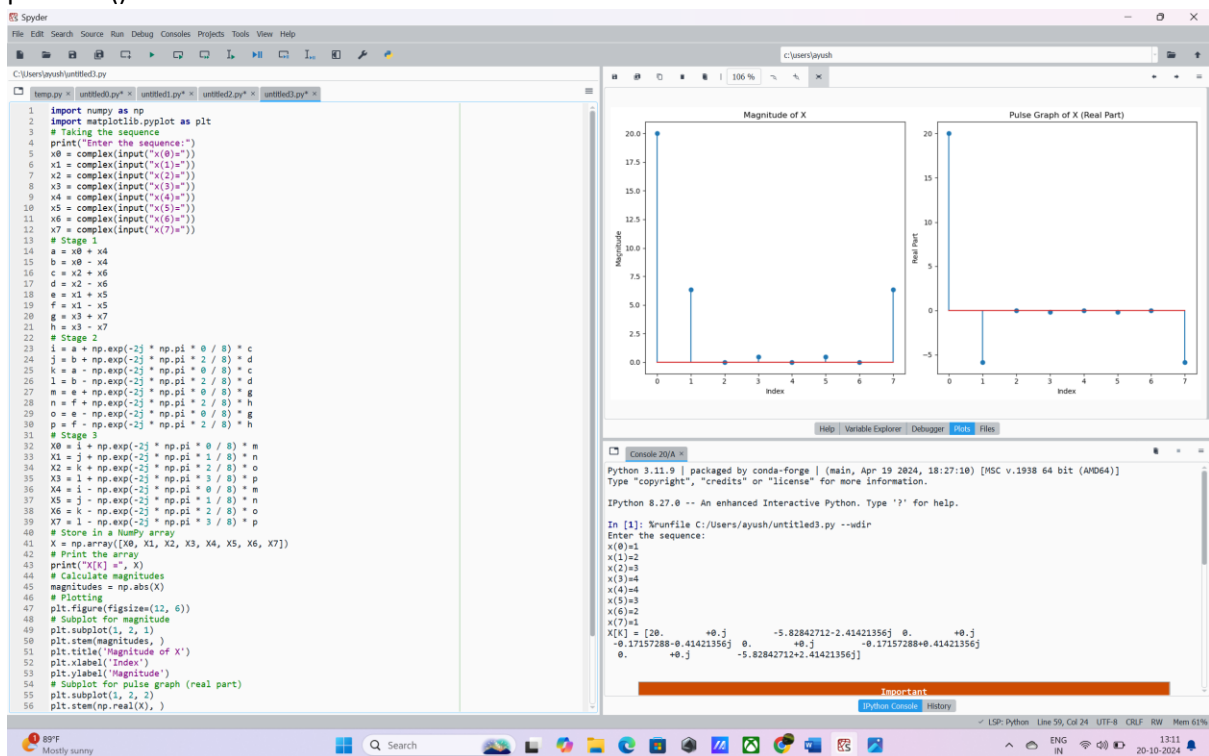
```
# Calculate magnitudes
magnitudes = np.abs(X)
```

```
# Plotting
plt.figure(figsize=(12, 6))
```

```
# Subplot for magnitude
plt.subplot(1, 2, 1)
plt.stem(magnitudes, )
plt.title('Magnitude of X')
plt.xlabel('Index')
plt.ylabel('Magnitude')
```

```
# Subplot for pulse graph (real part)
plt.subplot(1, 2, 2)
plt.stem(np.real(X), )
plt.title('Pulse Graph of X (Real Part)')
plt.xlabel('Index')
plt.ylabel('Real Part')
```

```
plt.tight_layout()
plt.show()
```



IDFT Using DIT FFT EXP-3

IDFT Using DIT FFT EXP-3

import numpy as np

import matplotlib.pyplot as plt

Taking the sequence

print("Enter the sequence:")

x0 = complex(input("x(0)="))

x1 = complex(input("x(1)="))

x2 = complex(input("x(2)="))

x3 = complex(input("x(3)="))

x4 = complex(input("x(4)="))

x5 = complex(input("x(5)="))

x6 = complex(input("x(6)="))

x7 = complex(input("x(7)="))

Stage 1

a = x0 + x4

b = x0 - x4

c = x2 + x6

d = x2 - x6

e = x1 + x5

f = x1 - x5

g = x3 + x7

h = x3 - x7

Stage 2

i = a + np.exp(-2j * np.pi * 0 / 8) * c

j = b + np.exp(-2j * np.pi * 2 / 8) * d

k = a - np.exp(-2j * np.pi * 0 / 8) * c

l = b - np.exp(-2j * np.pi * 2 / 8) * d

m = e + np.exp(-2j * np.pi * 0 / 8) * g

n = f + np.exp(-2j * np.pi * 2 / 8) * h

o = e - np.exp(-2j * np.pi * 0 / 8) * g

p = f - np.exp(-2j * np.pi * 2 / 8) * h

Stage 3

X0 = i + np.exp(-2j * np.pi * 0 / 8) * m

X1 = j + np.exp(-2j * np.pi * 1 / 8) * n

X2 = k + np.exp(-2j * np.pi * 2 / 8) * o

X3 = l + np.exp(-2j * np.pi * 3 / 8) * p

X4 = i - np.exp(-2j * np.pi * 0 / 8) * m

X5 = j - np.exp(-2j * np.pi * 1 / 8) * n

X6 = k - np.exp(-2j * np.pi * 2 / 8) * o

X7 = l - np.exp(-2j * np.pi * 3 / 8) * p

Store in a NumPy array

X = np.array([X0, X1, X2, X3, X4, X5, X6, X7])

Print the array

print("X[K] =", X)

Calculate magnitudes

magnitudes = np.abs(X)

Subplot for magnitude

```
plt.subplot(1, 2, 1)
```

```
plt.stem(magnitudes) # Removed use_line_collection=True
```

```
plt.title('Magnitude of X')
```

```
plt.xlabel('Index')
```

```
plt.ylabel('Magnitude')
```

Subplot for pulse graph (real part)

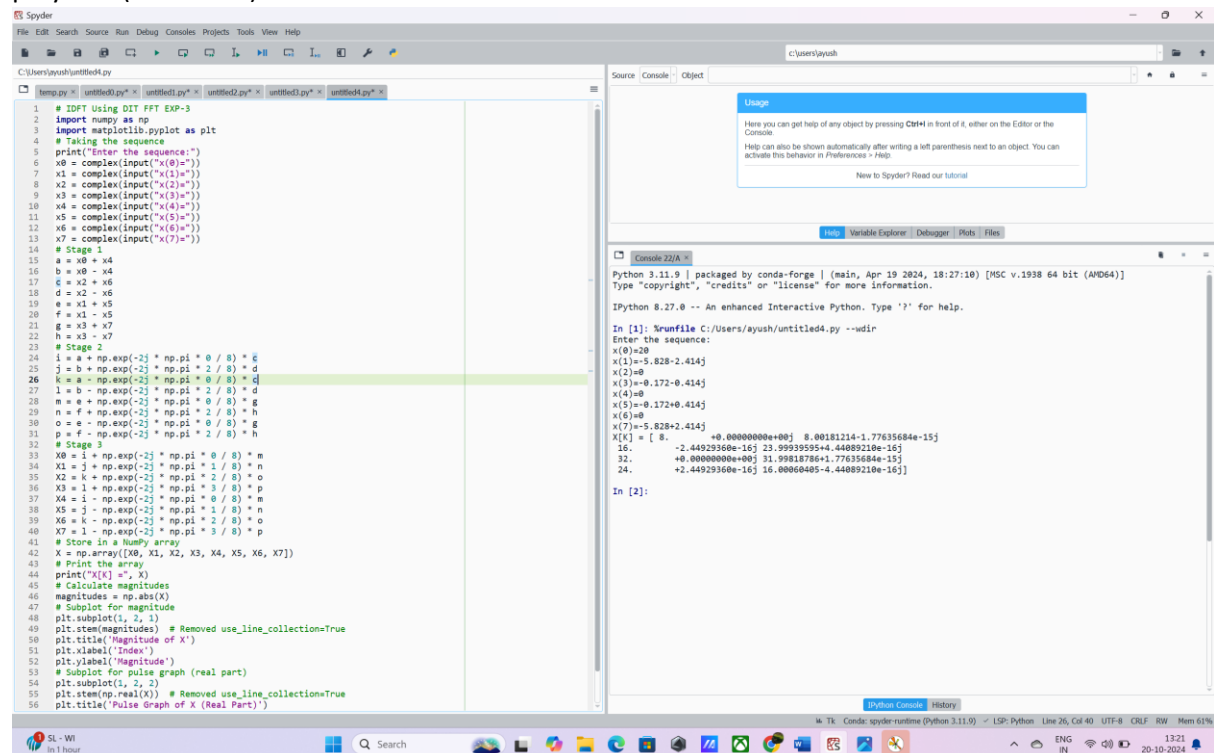
```
plt.subplot(1, 2, 2)
```

```
plt.stem(np.real(X)) # Removed use_line_collection=True
```

```
plt.title('Pulse Graph of X (Real Part)')
```

```
plt.xlabel('Index')
```

```
plt.ylabel('Real Part')
```



```
1 # IDFT Using DFT FFT EXP-3
2 import numpy as np
3 import matplotlib.pyplot as plt
4 # Taking the sequence
5 print("Enter the sequence:")
6 x0 = complex(input("x(0)="))
7 x1 = complex(input("x(1)="))
8 x2 = complex(input("x(2)="))
9 x3 = complex(input("x(3)="))
10 x4 = complex(input("x(4)="))
11 x5 = complex(input("x(5)="))
12 x6 = complex(input("x(6)="))
13 x7 = complex(input("x(7)="))
14 # Stage 1
15 a = x0 + x4
16 b = x0 - x4
17 c = x2 + x6
18 d = x2 - x6
19 e = x1 + x5
20 f = x1 - x5
21 g = x3 + x7
22 h = x3 - x7
23 # Stage 2
24 i = a + np.exp(-2j * np.pi * 0 / 8) * e
25 j = b + np.exp(-2j * np.pi * 2 / 8) * d
26 k = a - np.exp(-2j * np.pi * 0 / 8) * e
27 l = b - np.exp(-2j * np.pi * 2 / 8) * d
28 m = e + np.exp(-2j * np.pi * 0 / 8) * g
29 n = f + np.exp(-2j * np.pi * 2 / 8) * h
30 o = e - np.exp(-2j * np.pi * 0 / 8) * g
31 p = f - np.exp(-2j * np.pi * 2 / 8) * h
32 # Stage 3
33 x0 = i + np.exp(-2j * np.pi * 0 / 8) * m
34 x1 = j + np.exp(-2j * np.pi * 1 / 8) * n
35 x2 = k + np.exp(-2j * np.pi * 2 / 8) * o
36 x3 = l + np.exp(-2j * np.pi * 3 / 8) * p
37 x4 = i - np.exp(-2j * np.pi * 0 / 8) * m
38 x5 = j - np.exp(-2j * np.pi * 1 / 8) * n
39 x6 = k - np.exp(-2j * np.pi * 2 / 8) * o
40 x7 = l - np.exp(-2j * np.pi * 3 / 8) * p
41 # Store in a NumPy array
42 X = np.array([x0, x1, x2, x3, x4, x5, x6, x7])
43 # Print the array
44 print("X[k] = ", X)
45 # Calculate magnitudes
46 magnitudes = np.abs(X)
47 # Subplot for magnitude
48 plt.subplot(1, 2, 1)
49 plt.stem(magnitudes) # Removed use_line_collection=True
50 plt.title('Magnitude of X')
51 plt.xlabel('Index')
52 plt.ylabel('Magnitude')
53 # Subplot for pulse graph (real part)
54 plt.subplot(1, 2, 2)
55 plt.stem(np.real(X)) # Removed use_line_collection=True
56 plt.title('Pulse Graph of X (Real Part)')
```

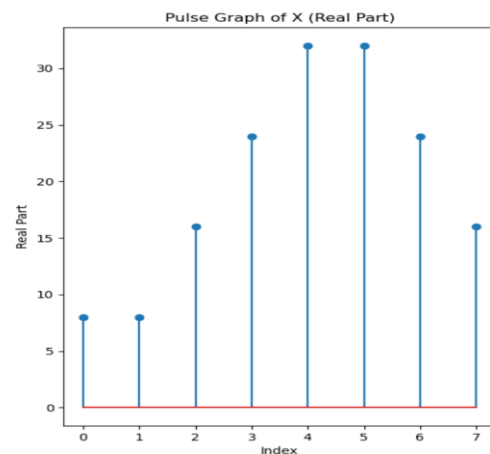
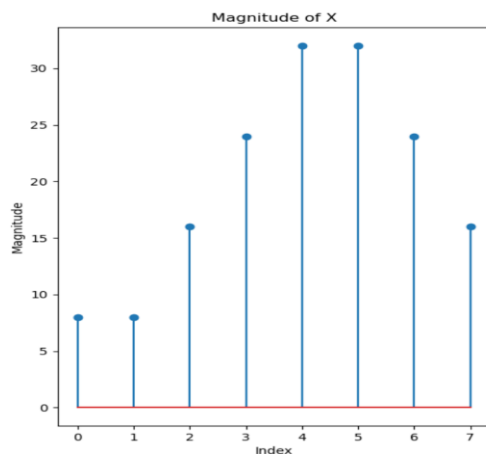
Console 22/A

```
Python 3.11.9 | packaged by conda-forge | (main, Apr 19 2024, 18:27:10) [MSC v.1938 64 bit (AMD64)]
Type "copyright", "credits" or "license()" for more information.

IPython 8.27.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: %runfile C:/Users/ayush/untitled4.py --wdir
Enter the sequence:
x(0)=0
x(1)=-5.828-2.414j
x(2)=0
x(3)=0.172-0.414j
x(4)=0
x(5)=0.172+0.414j
x(6)=0
x(7)=-5.828+2.414j
X[k] = [ 8. +0.00000000e+00j  8.00181214-1.77635684e-15j
 16. -2.44929360e-16j  23.99939595+4.44892100e-16j
 32. +0.00000000e+00j  31.99818786+1.77635684e-15j
 24. -2.44929360e-16j  16.00060485-4.44892100e-16j]

In [2]:
```



Take numerator and denominator from user Exp-4

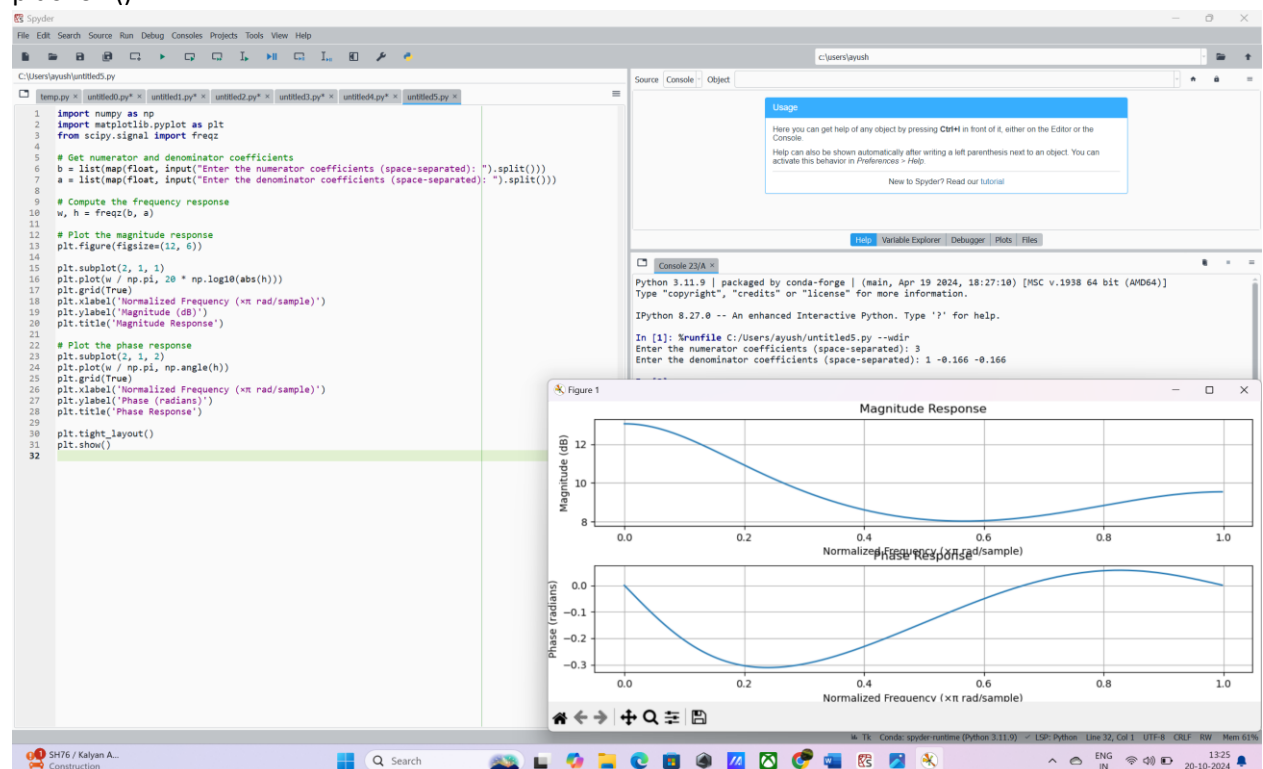
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import freqz

# Get numerator and denominator coefficients
b = list(map(float, input("Enter the numerator coefficients (space-separated): ").split()))
a = list(map(float, input("Enter the denominator coefficients (space-separated): ").split()))

# Compute the frequency response
w, h = freqz(b, a)

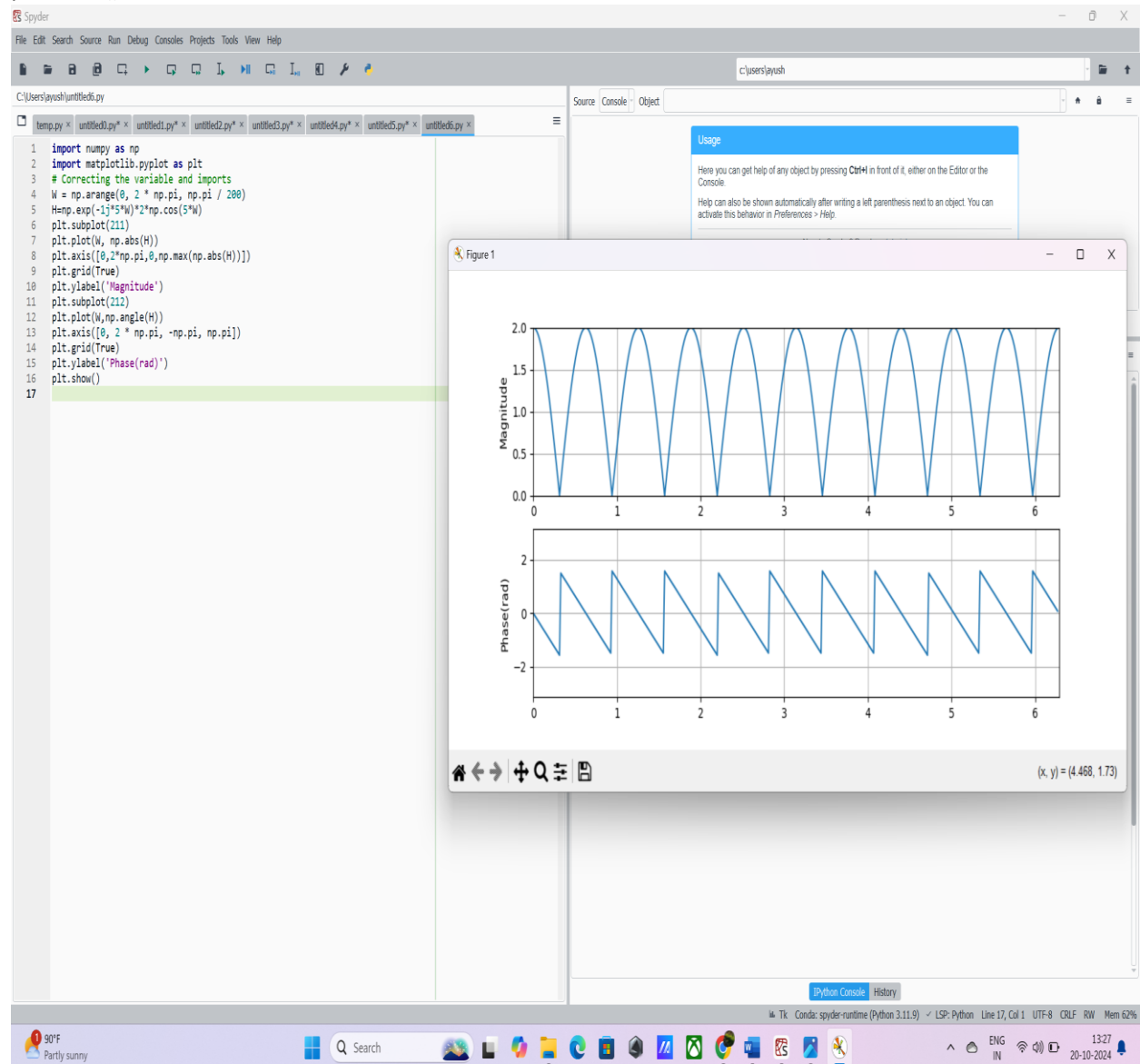
# Plot the magnitude response
plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
plt.plot(w / np.pi, 20 * np.log10(abs(h)))
plt.grid(True)
plt.xlabel('Normalized Frequency ( $\times\pi$  rad/sample)')
plt.ylabel('Magnitude (dB)')
plt.title('Magnitude Response')

# Plot the phase response
plt.subplot(2, 1, 2)
plt.plot(w / np.pi, np.angle(h))
plt.grid(True)
plt.xlabel('Normalized Frequency ( $\times\pi$  rad/sample)')
plt.ylabel('Phase (radians)')
plt.title('Phase Response')
plt.tight_layout()
plt.show()
```



finding /compute the sequence and sketch amplitude/phase

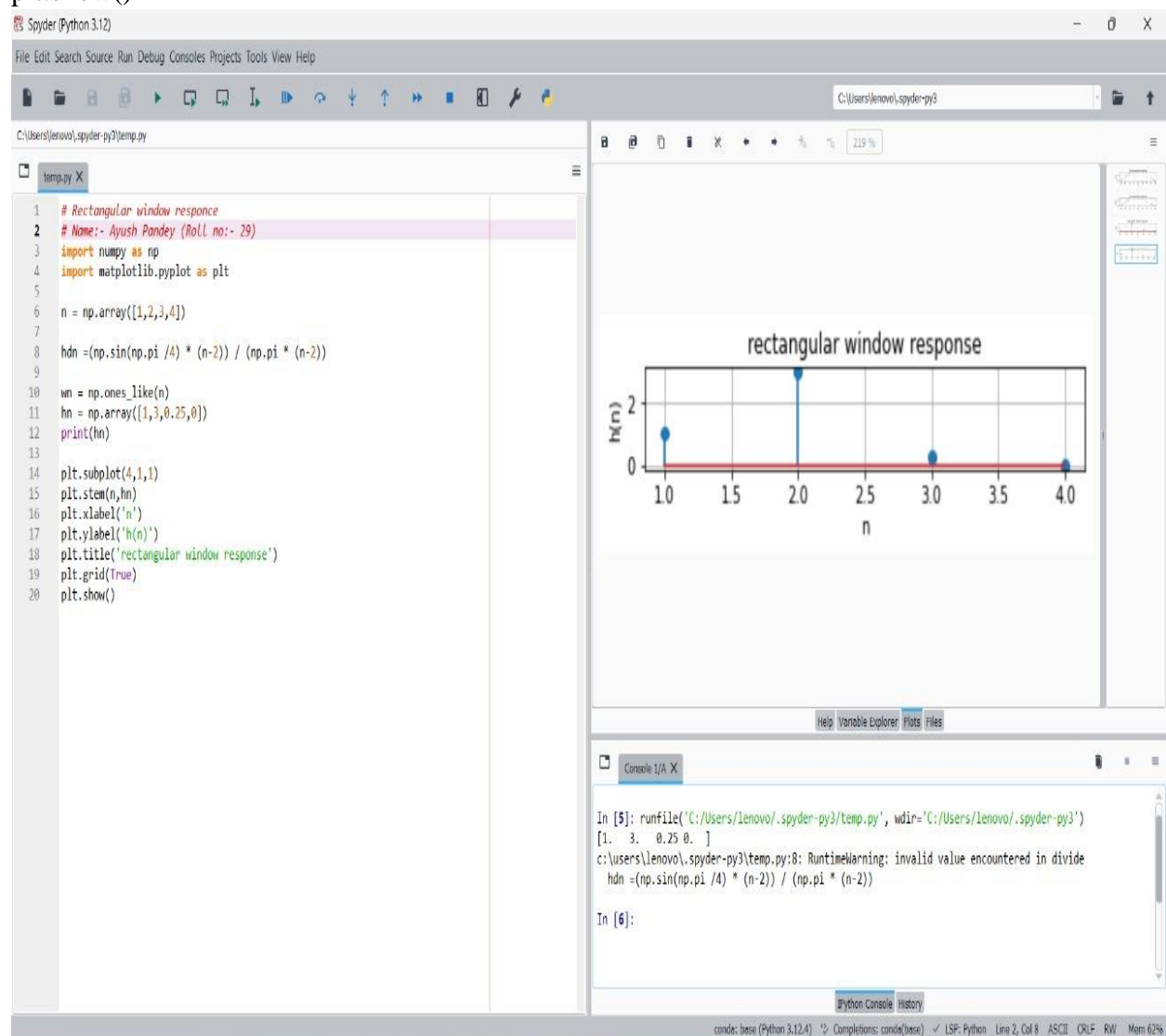
```
import numpy as np
import matplotlib.pyplot as plt
# Correcting the variable and imports
W = np.arange(0, 2 * np.pi, np.pi / 200)
H=np.exp(-1j*5*W)*2*np.cos(5*W)
plt.subplot(211)
plt.plot(W, np.abs(H))
plt.axis([0,2*np.pi,0,np.max(np.abs(H))])
plt.grid(True)
plt.ylabel('Magnitude')
plt.subplot(212)
plt.plot(W,np.angle(H))
plt.axis([0, 2 * np.pi, -np.pi, np.pi])
plt.grid(True)
plt.ylabel('Phase(rad)')
plt.show()
```



EXP-05

```
import numpy as np #Rectangular window
import matplotlib.pyplot as plt
n= np.array([1,2,3,4])
hdn = (np.sin(np.pi/4) * (n-2)) / (np.pi * (n-2))
wn = np.ones_like(n)
hn = np.array([1,3,0.25,0])
print(hn)
plt.subplot(4,1,1)
plt.stem(n,hn)
plt.xlabel('n')
plt.ylabel('h(n)')
plt.title('Rectangular window response')
plt.grid(True)

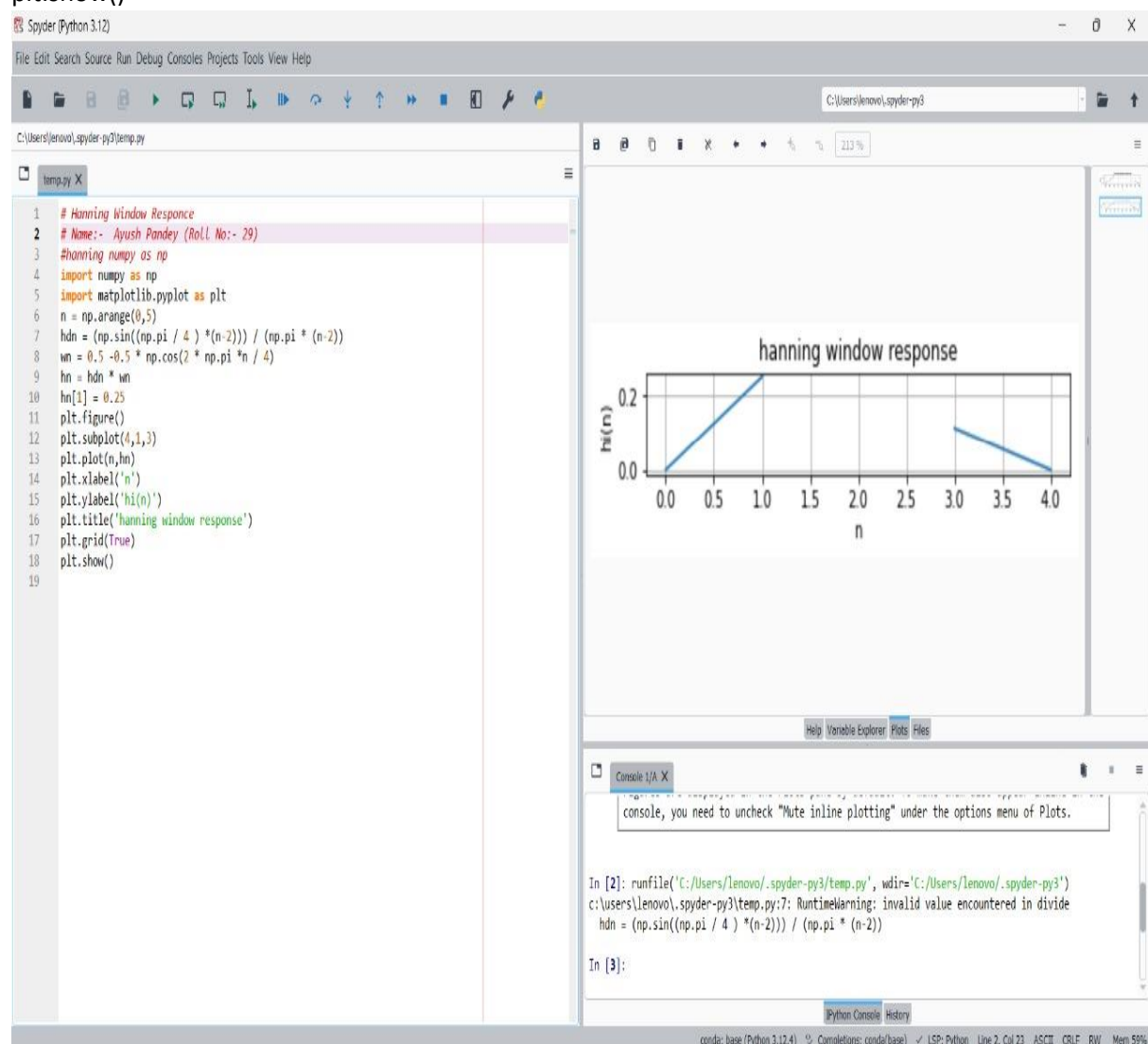
plt.show()
```



```

import numpy as np
import numpy as np #Hanning window
import matplotlib.pyplot as plt
n = np.arange(0, 5)
hdn = (np.sin((np.pi / 4) * (n - 2))) / (np.pi * (n - 2))
wn = 0.5 - 0.5 * np.cos(2 * np.pi * n / 4)
hn = hdn * wn
hn[1] = 0.25
plt.figure()
plt.subplot(4,1,3)
plt.plot(n, hn)
plt.xlabel('n')
plt.ylabel('hi(n)')
plt.title('Hanning Window Response')
plt.grid(True)
plt.show()

```

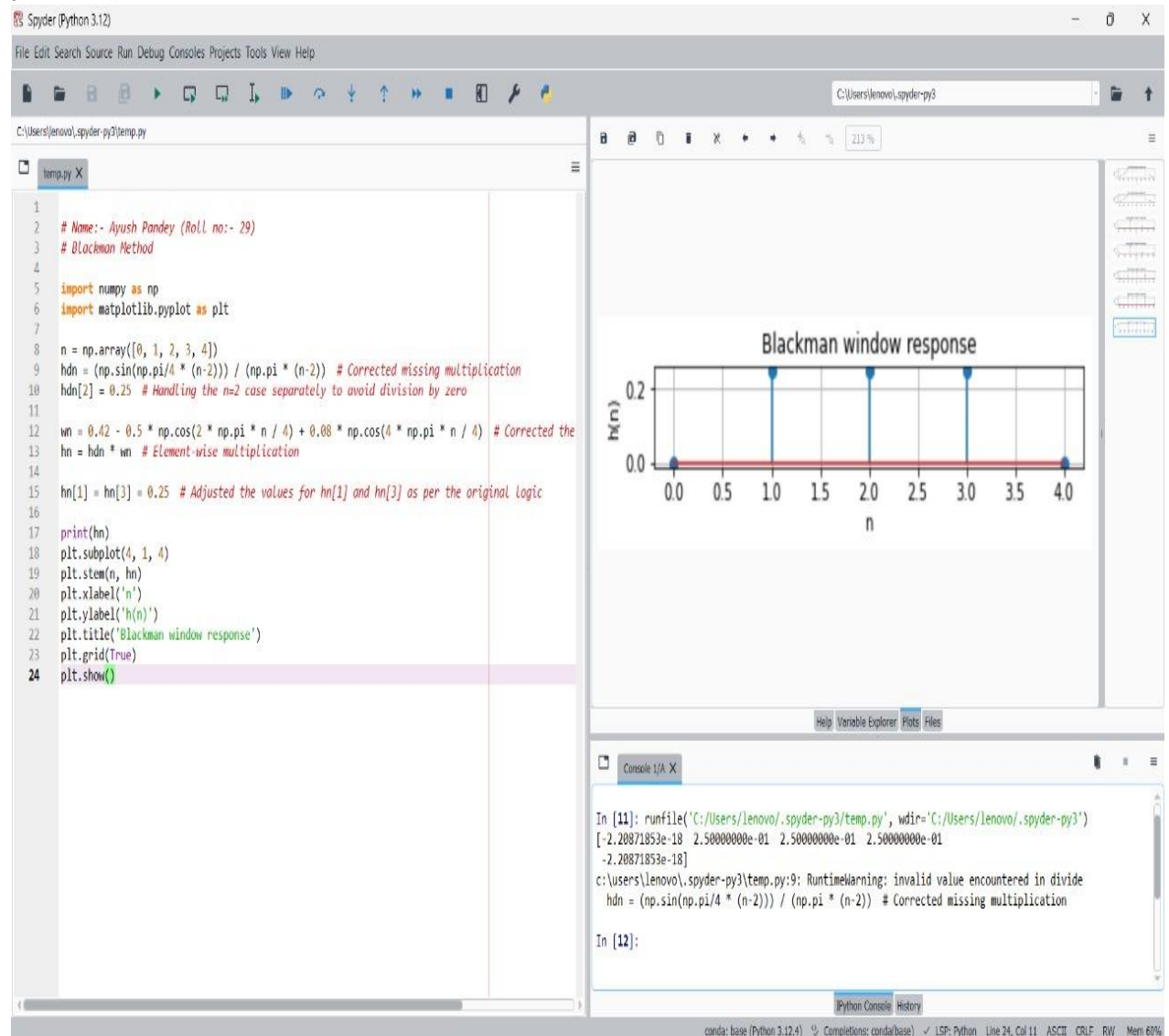


```

import numpy as np #Blackam window
import matplotlib.pyplot as plt
n = np.array([0,1,2,3,4])
hdn = (np.sin(np.pi / 4) * (n - 2)) / (np.pi * (n - 2))
hdn[2]=0.25
wn = 0.42-0.5*np.cos(2* np.pi*n/ 4)+0.08*np.cos(4*np.pi*n/4)
hn = hdn* wn
hn[1]=hn[3]=0.25

print(hn)
plt.subplot(4,1,2)
plt.stem(n,hn)
plt.xlabel('n')
plt.ylabel('h(n)')
plt.title('Hamming window response')
plt.grid(True)
plt.show()

```

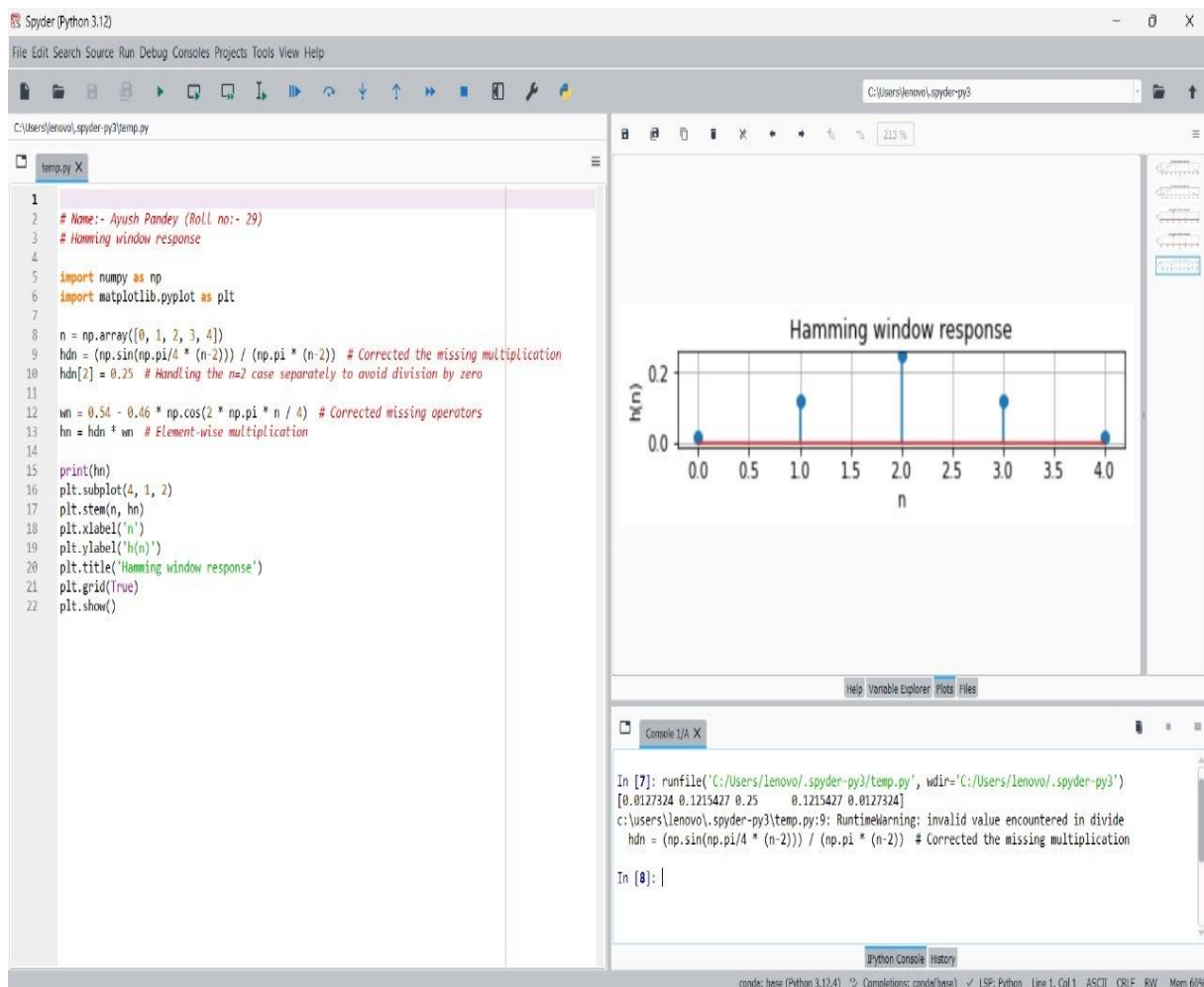


```

import numpy as np #Hamming Window
import matplotlib.pyplot as plt
n = np.array([0,1,2,3,4])
hdn = (np.sin(np.pi / 4) * (n - 2)) / (np.pi * (n - 2))
hdn[2]=0.25
wn = 0.54 - 0.46*np.cos(2* np.pi*n/ 4)
hn = hdn* wn

print(hn)
plt.subplot(4,1,2)
plt.stem(n,hn)
plt.xlabel('n')
plt.ylabel('h(n)')
plt.title('Hamming window response')
plt.grid(True)
plt.show()

```



Experiment no 6

#To design and implement a high pass finite impulse response for a sequence

import numpy as np

for n in range(1,6):

 hdn=(1/(np.pi*n))*(np.sin(np.pi*n)-np.sin((np.pi*n)/4))

 wn=(0.54+0.46*np.cos(np.pi*n/5))

 hn=wn*hdn

 print(np.array(hn))

The screenshot displays the Spyder Python IDE interface. The left pane shows a script named 'untitled0.py' with the following code:

```
1 #exp-6 ayush pandey-29
2 #To design and implement a high pass finite impulse response for a sequence
3 import numpy as np
4 for n in range(1,6):
5     hdn=(1/(np.pi*n))*(np.sin(np.pi*n)-np.sin((np.pi*n)/4))
6     wn=(0.54+0.46*np.cos(np.pi*n/5))
7     hn=wn*hdn
8     print(np.array(hn))
```

The right pane features a Variable Explorer showing the values of variables defined in the script:

Name	Type	Size	Value
hdn	Float64	1	np.float64(0.04581581580785346)
hn	Float64	1	np.float64(0.0036012652646284283)
n	int	1	5
wn	Float64	1	np.float64(0.6800000000000002)

Below the Variable Explorer is the IPython Console, which shows the execution output:

```
Python 3.11.9 | packaged by conda-forge | (main, Apr 19 2024, 18:27:18) [MSC v.1938 64 bit (AMD64)]
Type "copyright", "credits" or "license()" for more information.

IPython 8.27.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: %runfile C:/Users/ayush/untitled0.py --wdir
+0.2653853986988852
+0.10856719706054323
+0.029849408950188672
+0.17895822736983e-18
0.0036012652646284283

In [2]:
```

The bottom status bar indicates the system is at 88°F, mostly sunny, with a search bar and system icons on the right.

#IIR filter Experiment 7

```
import numpy as np
from scipy import signal
ohm=4
wc=np.pi/2
T=2*np.tan(wc/2)/ohm
fs=1/T
n=np.array([1,0.1])
d=np.array([1,0.2,16.01])
Nr,Dr=signal.bilinear(n,d,fs)
print("Nr:",Nr, "\nDr:",Dr)
```

The screenshot displays the Spyder Python IDE interface. The main editor window on the left contains a Python script for an IIR filter experiment. The script defines parameters for a bilinear filter, including sampling frequency, cutoff frequency, and filter coefficients, and then uses the `signal.bilinear` function to compute the numerator and denominator coefficients. The results are printed to the console.

The right-hand side of the IDE is divided into two panels. The top panel, labeled 'Console', shows the output of the script execution, displaying the calculated values for `Nr` and `Dr`. The bottom panel, labeled 'Python Console', shows the IPython prompt and the execution of the script, including the file path and the output of the `print` statement.

The bottom status bar of the IDE indicates the current file is `untitled7.py`, the Python version is 3.11.9, and the environment is Conda: spyder-runtime (Python 3.11.9). The system tray at the very bottom shows the date and time as 20-10-2024, 13:57.

#part 2

```
import numpy as np
from scipy import signal
ohm=4
wc=np.pi/2
T=2*np.tan(wc/2)/ohm
fs=1/T
n=np.array([1,0.1])
d=np.array([1,0.2,9.01])
Nr,Dr=signal.bilinear(n,d,fs)
print("Nr:",Nr, "\nDr:",Dr)
```

The screenshot shows the Spyder Python IDE interface. The top bar indicates the system time as Mon 10:57 and the application as Spyder (Python 3.6). The menu bar includes File, Edit, Search, Source, Run, Debug, Consoles, Projects, Tools, View, and Help. The toolbar contains icons for file operations, running, and debugging. The editor window displays a Python script for IIR filter design. The IPython console on the right shows the execution output, including version information and the numerical results for Nr and Dr. The status bar at the bottom provides details on permissions, line endings, encoding, and current cursor position.

```
1 # IIR Filter design Ayush Pandey Roll no:-29
2 import numpy as np
3 from scipy import signal
4 ohm=4
5 wc=np.pi/2
6 T=2*np.tan(wc/2)/ohm
7 fs=1/T
8 n=np.array([1,0.1])
9 d=np.array([1,0.2,9.01])
10 Nr,Dr=signal.bilinear(n,d,fs)
11 print("Nr:",Nr, "\nDr:",Dr)
```

Python 3.6.9 (default, Jun 29 2022, 11:45:57)
Type "copyright", "credits" or "license" for more information.

IPython 5.5.0 -- An enhanced Interactive Python.
? -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help -> Python's own help system.
object? -> Details about 'object', use 'object??' for extra details.

In [1]: runfile('/home/computer/untitled12.py', wdir='/home/computer')
nr [0.15885316 0.08774893 -0.15110422]
dr [1. -0.54165052 0.93800852]

In [2]:

Permissions: RW End-of-lines: LF Encoding: UTF-8 Line: 1 Column: 45 Memory: 36 %

#Experiment 8 To design & Implemented High pass IIR filter

```
import numpy as np
from scipy.signal import butter, lfilter
import matplotlib.pyplot as plt

# -----
# 1. Define the Input Sequence
# -----
input_sequence = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# -----
# 2. Define Filter Parameters
# -----
Fs = 1000 # Sampling Frequency in Hz
Fc = 50   # Cutoff Frequency in Hz
N = 4     # Filter Order

# -----
# 3. Design the High-Pass Butterworth Filter
# -----
# Normalize the cutoff frequency with respect to Nyquist Frequency
Wn = Fc / (Fs / 2) # Normalized cutoff frequency (0 < Wn < 1)

# Get filter coefficients
b, a = butter(N, Wn, btype='high', analog=False)

# -----
# 4. Apply the Filter to the Input Sequence
# -----
filtered_sequence = lfilter(b, a, input_sequence)

# -----
# 5. Prepare Time Axis for Plotting
# -----
# Assuming each sample is taken at intervals of 1/Fs seconds
t = np.arange(len(input_sequence)) / Fs # Time axis in seconds

# -----
# 6. Plot the Original and Filtered Sequences
# -----
plt.figure(figsize=(12, 8))

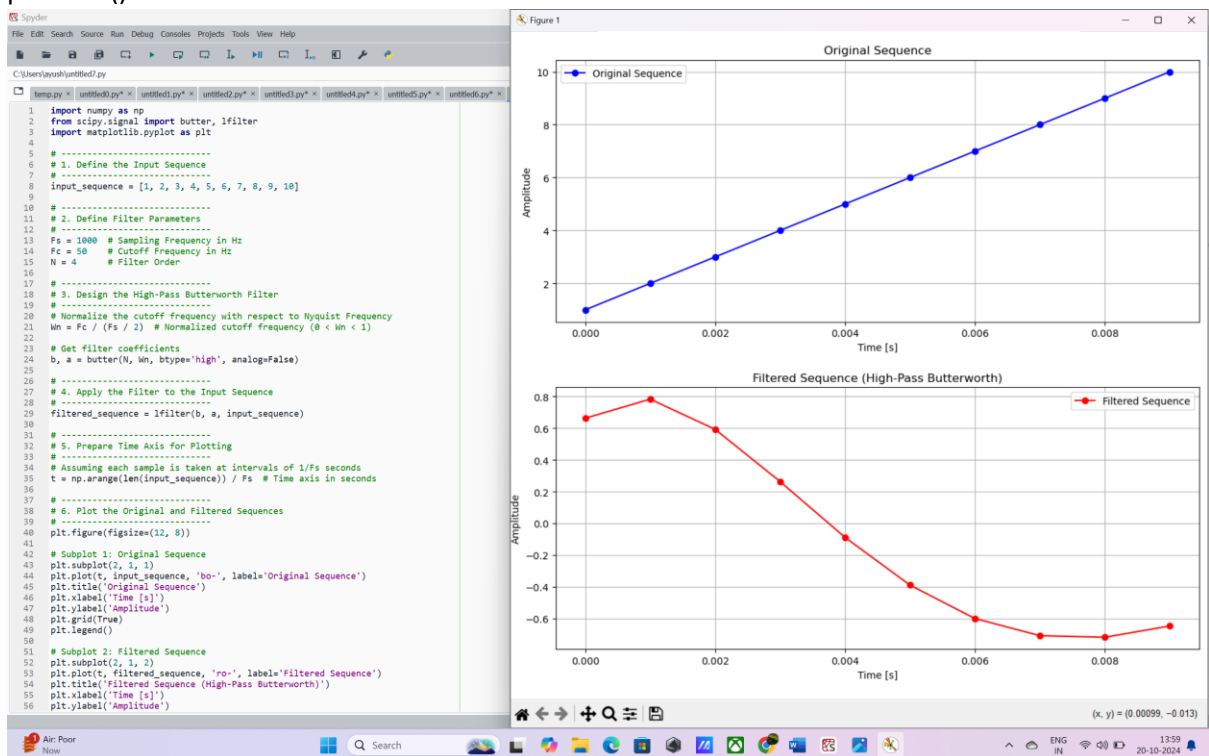
# Subplot 1: Original Sequence
plt.subplot(2, 1, 1)
plt.plot(t, input_sequence, 'bo-', label='Original Sequence')
plt.title('Original Sequence')
plt.xlabel('Time [s]')
plt.ylabel('Amplitude')
```

```
plt.grid(True)
plt.legend()
```

```
# Subplot 2: Filtered Sequence
```

```
plt.subplot(2, 1, 2)
plt.plot(t, filtered_sequence, 'ro-', label='Filtered Sequence')
plt.title('Filtered Sequence (High-Pass Butterworth)')
plt.xlabel('Time [s]')
plt.ylabel('Amplitude')
plt.grid(True)
plt.legend()
```

```
plt.tight_layout()
plt.show()
```



Experiment 9 Decimation Process

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

# Define the original signal
xn = np.array([1, -1, 1, -1, 2, -2, 2, -2, 3, -3, 3, -3])
N = len(xn)
n = np.arange(N)

# Downsample by factor of 2
xD2n = xn[::2]
n1 = np.arange(len(xD2n))

# Downsample by factor of 3
xD3n = xn[::3]
n2 = np.arange(len(xD3n))

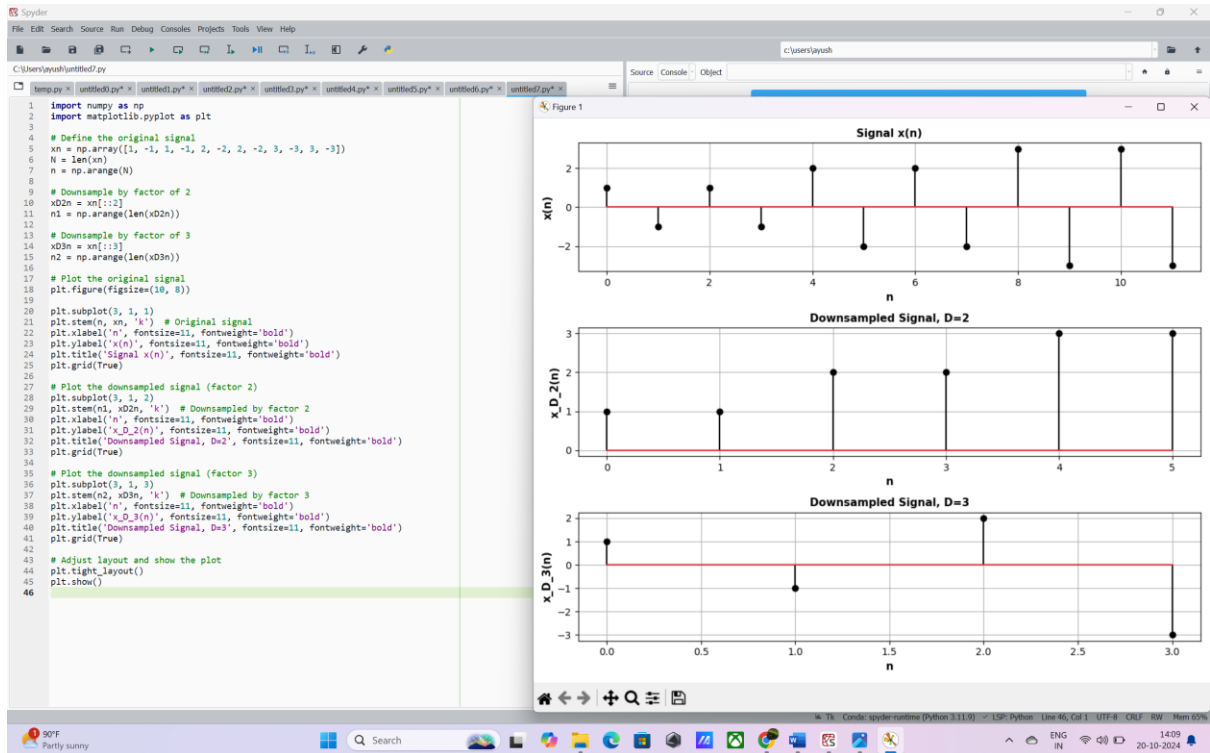
# Plot the original signal
plt.figure(figsize=(10, 8))

plt.subplot(3, 1, 1)
plt.stem(n, xn, 'k') # Original signal
plt.xlabel('n', fontsize=11, fontweight='bold')
plt.ylabel('x(n)', fontsize=11, fontweight='bold')
plt.title('Signal x(n)', fontsize=11, fontweight='bold')
plt.grid(True)

# Plot the downsampled signal (factor 2)
plt.subplot(3, 1, 2)
plt.stem(n1, xD2n, 'k') # Downsampled by factor 2
plt.xlabel('n', fontsize=11, fontweight='bold')
plt.ylabel('x_D_2(n)', fontsize=11, fontweight='bold')
plt.title('Downsampled Signal, D=2', fontsize=11, fontweight='bold')
plt.grid(True)

# Plot the downsampled signal (factor 3)
plt.subplot(3, 1, 3)
plt.stem(n2, xD3n, 'k') # Downsampled by factor 3
plt.xlabel('n', fontsize=11, fontweight='bold')
plt.ylabel('x_D_3(n)', fontsize=11, fontweight='bold')
plt.title('Downsampled Signal, D=3', fontsize=11, fontweight='bold')
plt.grid(True)

# Adjust layout and show the plot
plt.tight_layout()
plt.show()
```



Experiment 10 understanding how to read sound file

