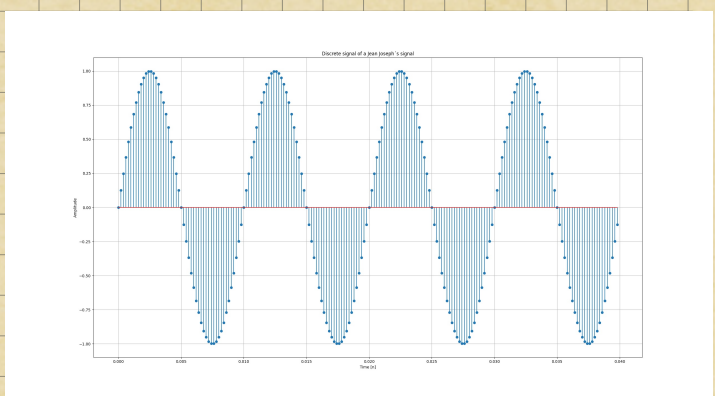
Task 1)

Code:

Plot:

Øving 2 > Oppgave1.py > ...

```python
#Task 1
import numpy as np
import matplotlib.pyplot as plt
#Defineng some variables
f = 100
A = 1
samplingTime = 0.2*10**-3
samplingRate = 1/(samplingTime)

#Defining the time vector
NumberOfSamples = 900
t = np.linspace(0,NumberOfSamples*(samplingTime),NumberOfSamples+1)
#Defining the signal
signal = A*np.sin(2*np.pi*f*t)
#plotting the signal
#Just want to plot the first 200 values
plt.figure(1)
plt.stem(t[0:200],signal[0:200])
plt.xlabel('Time [n]')
plt.ylabel('Amplitude')
plt.title('Discrete signal of a Jean Joseph´s signal')
plt.grid(True)
plt.show()
```



$$\text{Samplingfrekvens} = \frac{1}{0.2 \cdot 10^{-3}} = \underline{5000\,Hz}$$

Nyquist-grensen blir den doble verdien til den høyeste frekvensen til Signalet. Dette blir altså en verdi på $\underline{200\,Hz.}$

Task 2)

Code:

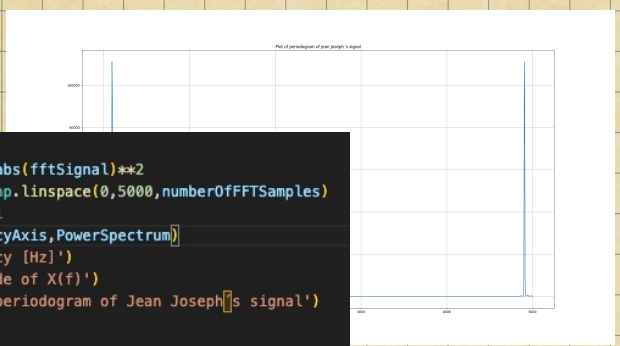
a)

```python
#Task 2
import numpy as np
import matplotlib.pyplot as plt
#Definng some variables
f = 100
A = 1
samplingTime = 0.2*10**-3
samplingRate = 1/(samplingTime)

#Defining the time vector
NumberOfSamples = 900
t = np.linspace(0,NumberOfSamples*(samplingTime),NumberOfSamples+1)
#Defining the signal
signal = A*np.sin(2*np.pi*f*t)
#Calculate the FFT of the signal
numberOfFFTSamples = 1024
#Frequency axis
frequency_step = samplingRate/numberOfFFTSamples
frequency_axis = np.fft.fftfreq(numberOfFFTSamples,samplingTime)
fftSignal = np.fft.fft(signal,numberOfFFTSamples)
#Convert to dB
ffTDesibelSignal = 20*np.log10(np.abs(fftSignal))
```

b)

Code

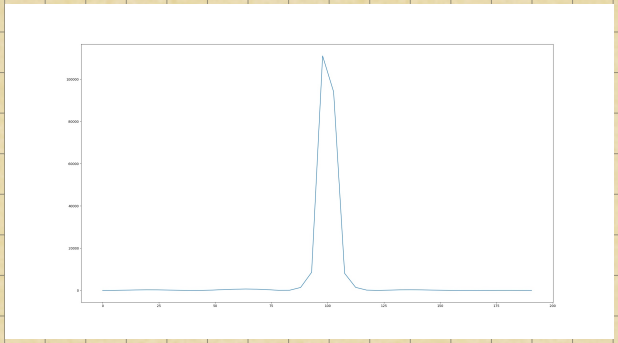
Plot



```python
#Task 2 b)
PowerSpectrum = np.abs(fftSignal)**2
newFrequencyAxis = np.linspace(0,5000,numberOfFFTSamples)
#plotting the signal
plt.plot(newFrequencyAxis,PowerSpectrum)
plt.xlabel('Frequency [Hz]')
plt.ylabel('Amplitude of X(f)')
plt.title('Plot of periodogram of Jean Joseph's signal')
plt.grid(True)
plt.show()
```

Aliasing frekvensen oppstår pga frekvensen 4900 Hz som forventet.

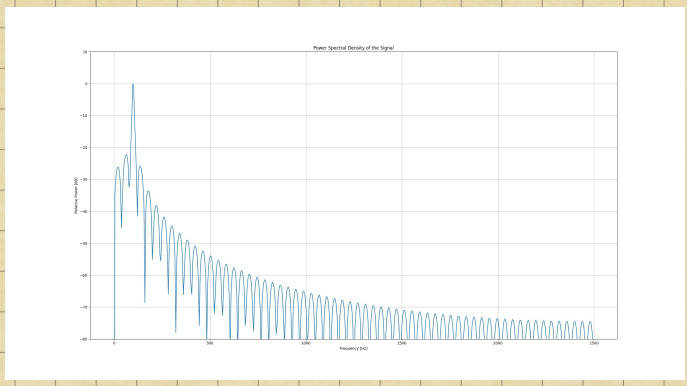
c)



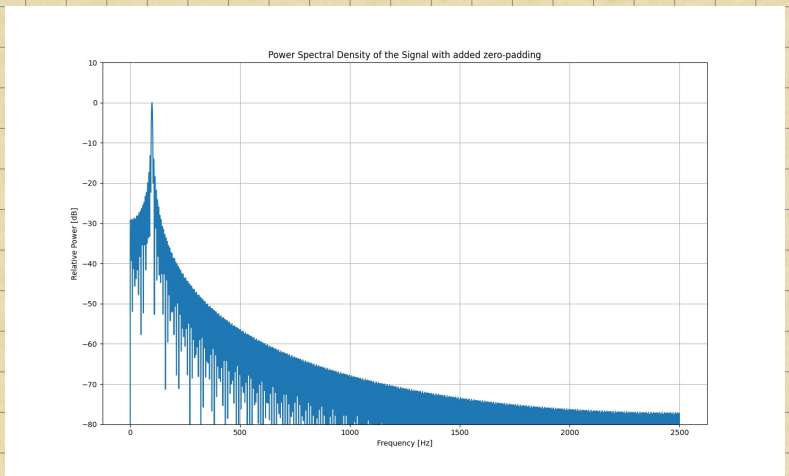
d) Code:

```python
#Task 2 d)

# Calculate Power Spectrum and Convert to dB and normalize
PowerSpectrum_dB = 20 * np.log10(np.abs(fftSignal) / np.sqrt(numberOfFFTSamples))
PowerSpectrum_dB = PowerSpectrum_dB - max(PowerSpectrum_dB)

# Plot the signal

plt.plot(frequency_axis[:numberOfFFTSamples // 2], PowerSpectrum_dB[:numberOfFFTSamples // 2])
plt.xlabel('Frequency [Hz]')
plt.ylabel('Relative Power [dB]')
plt.title('Power Spectral Density of the Signal')
plt.grid(True)
plt.ylim(-80, 10)  # Set y-axis limits
plt.show()
```

Result:

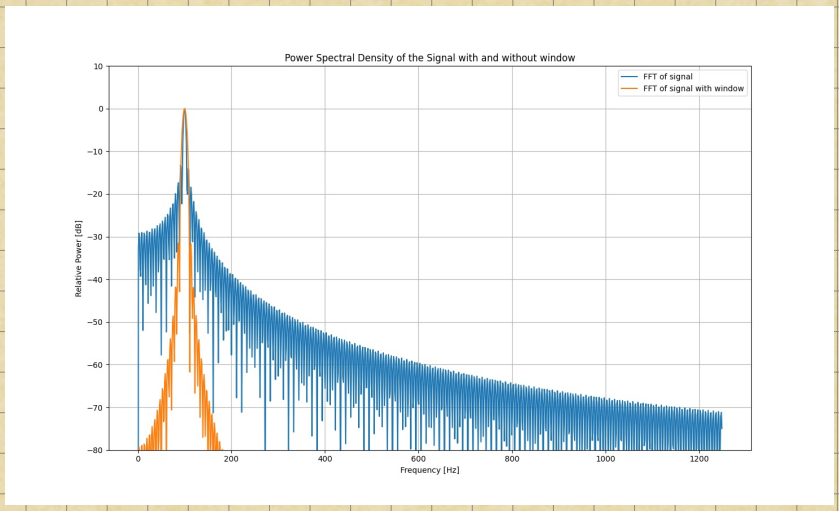


Task 3)




```
#We can clearly see that when we change the length of the FFT we change the distance of the frequency-step.
#This is even more clearer when we change to a length of 2*4096
```

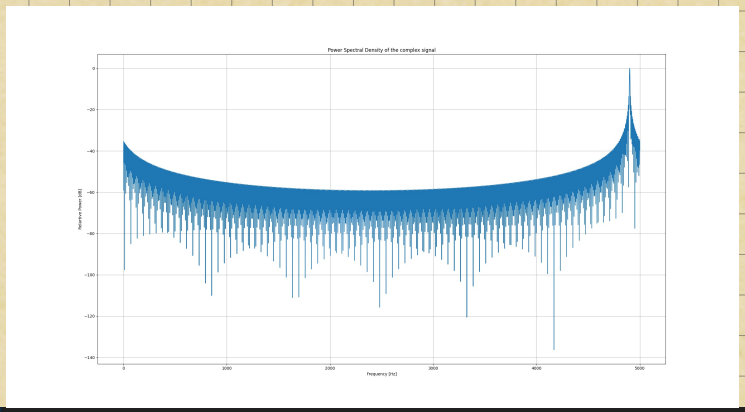
Task 4)




```python
#Same but with window-function

import numpy as np
import matplotlib.pyplot as plt

#Definng some variables
f = 100
A = 1
samplingTime = 0.2*10**-3
samplingRate = 1/(samplingTime)

#Defining the time vector
NumberOfSamples = 900
t = np.linspace(0,NumberOfSamples*(samplingTime),NumberOfSamples+1)
#Defining the signal
signal = A*np.sin(2*np.pi*f*t)
#Create a window function. We use the Hanning window
windowFunction = np.hanning(NumberOfSamples+1)
#Multiply the signal with the window function
windoSignal = signal*windowFunction
#Calculate the FFT of the signal
numberOfFFTSamples = 4096
#Frequency axis
frequency_step = samplingRate/numberOfFFTSamples
frequency_axis = np.fft.fftfreq(numberOfFFTSamples)
fftSignal = np.fft.fft(signal,numberOfFFTSamples)
windowFFTSignal = np.fft.fft(windoSignal,numberOfFFTSamples)
#Convert to dB
PowerSpectrum = np.abs(fftSignal)**2
windowPowerSpectrum = np.abs(windowFFTSignal)**2
newFrequencyAxis = np.linspace(0,5000,numberOfFFTSamples)
# Calculate Power Spectrum and Convert to dB and normalize
PowerSpectrum_dB = 20 * np.log10(np.abs(fftSignal) / np.sqrt(numberOfFFTSamples))
PowerSpectrum_dB = PowerSpectrum_dB - max(PowerSpectrum_dB)
#Nomalize the window-signal
windowPowerSpectrum_dB = 20 * np.log10(np.abs(windowFFTSignal) / np.sqrt(numberOfFFTSamples))
```

```
#As we can see using a Hanning-window function we get a lot less side lobes. This however comes at a price
#of a broder main lobe. This can make it harder to distinguish between frequencies close to each other.
#This is a trade-off that we have to make. We can also observe that the amplitude of the windowed signal is smoother.
#This is because the signal is less affected by the abrupt start and end points,
```

Task 5:



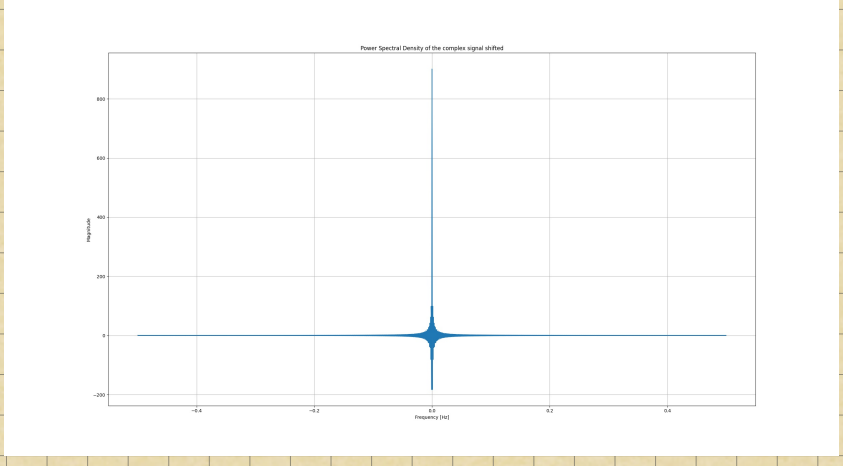
Oppgaven5.py > ...

```python
#Same but with complex signal
import numpy as np
import matplotlib.pyplot as plt
#Defining some variables
f = 100
A = 1
samplingTime = 0.2*10**-3
samplingRate = 1/(samplingTime)

#Defining the time vector
NumberOfSamples = 900
t = np.linspace(0,NumberOfSamples*(samplingTime),NumberOfSamples+1)
#Defining the signal
complexSignal = A*np.exp(-1j*2*np.pi*f*t)
#We need to find the PSD of the signal.
#Calculate the FFT of the signal
numberOfFFTSamples = 4096
#Frequency axis
frequency_axis = np.fft.fftfreq(numberOfFFTSamples)
newFrequencyAxis = np.linspace(0,samplingRate,numberOfFFTSamples)
fftComplexSignal = np.fft.fft(complexSignal,numberOfFFTSamples)

# Calculate Power Spectrum and Convert to dB and normalize
PowerSpectrum_dB = 20 * np.log10(np.abs(fftComplexSignal) / np.sqrt(numberOfFFTSamples))
PowerSpectrum_dB = PowerSpectrum_dB - max(PowerSpectrum_dB)
#plot the signal up to its sampling rate
plt.plot(newFrequencyAxis, PowerSpectrum_dB)
plt.xlabel('Frequency [Hz]')
plt.ylabel('Relartive Power [dB]')
plt.title('Power Spectral Density of the complex signal')
plt.grid(True)
plt.show()
```

```python
#We observe a peak at 4900 Hz. This is the alias of the signal at -100 Hz.
#This is because the signal is complex and not real. If we change the
#signal to complexSignal = A*np.exp(1j*2*np.pi*f*t) we get a REAL frequency
#that peaks at 100Hz.
```

b)

```python
#Task 5b)

#I have changed the frequency to 0 Hz
X_f_shitfted = np.fft.fftshift(fftComplexSignal)
freq = np.fft.fftshift(frequency_axis)
#Plot the shifted signal
plt.plot(freq, X_f_shitfted)
plt.xlabel('Frequency [Hz]')
plt.ylabel('Magnitude')
plt.title('Power Spectral Density of the complex signal shifted')
plt.grid(True)
plt.show()
```



c)

```
#Here we observe a peak at 100 Hz. This is because the signal now has a real frequency.
```