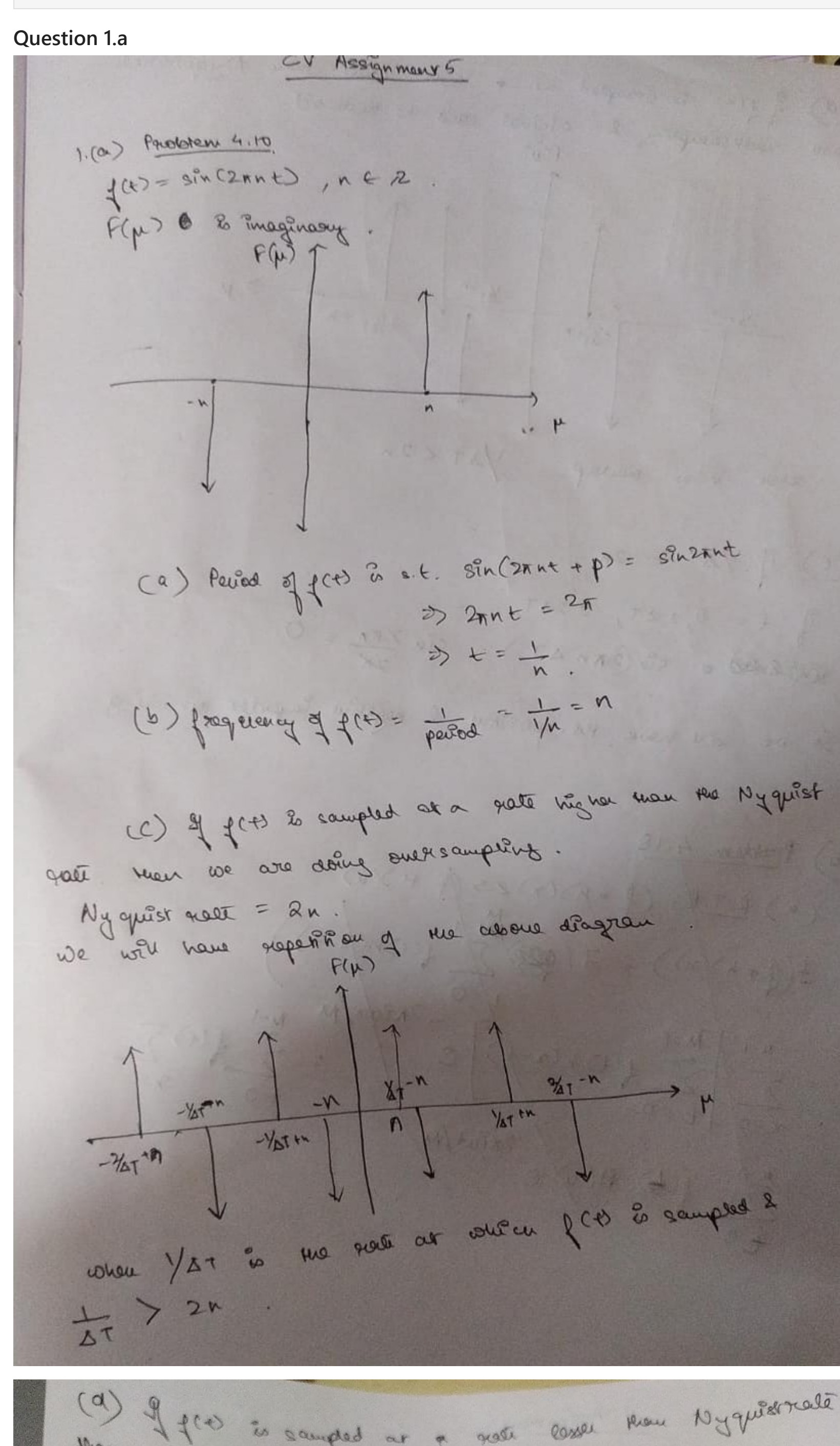
Question 1

In [9]:
```
#import the required libraries
import math
import cv2
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.signal import convolve2d
```

Question 1.a


CV Assignment 5

1.(a) Problem 4.10

$f(t) = \sin(2\pi n t)$, $n \in \mathbb{R}$

$F(\mu)$ is imaginary.

(a) Period of $f(t)$ is s.t. $\sin(2\pi n t + p) = \sin 2\pi n t \Rightarrow 2\pi n t = 2\pi \Rightarrow t = \frac{1}{n}$

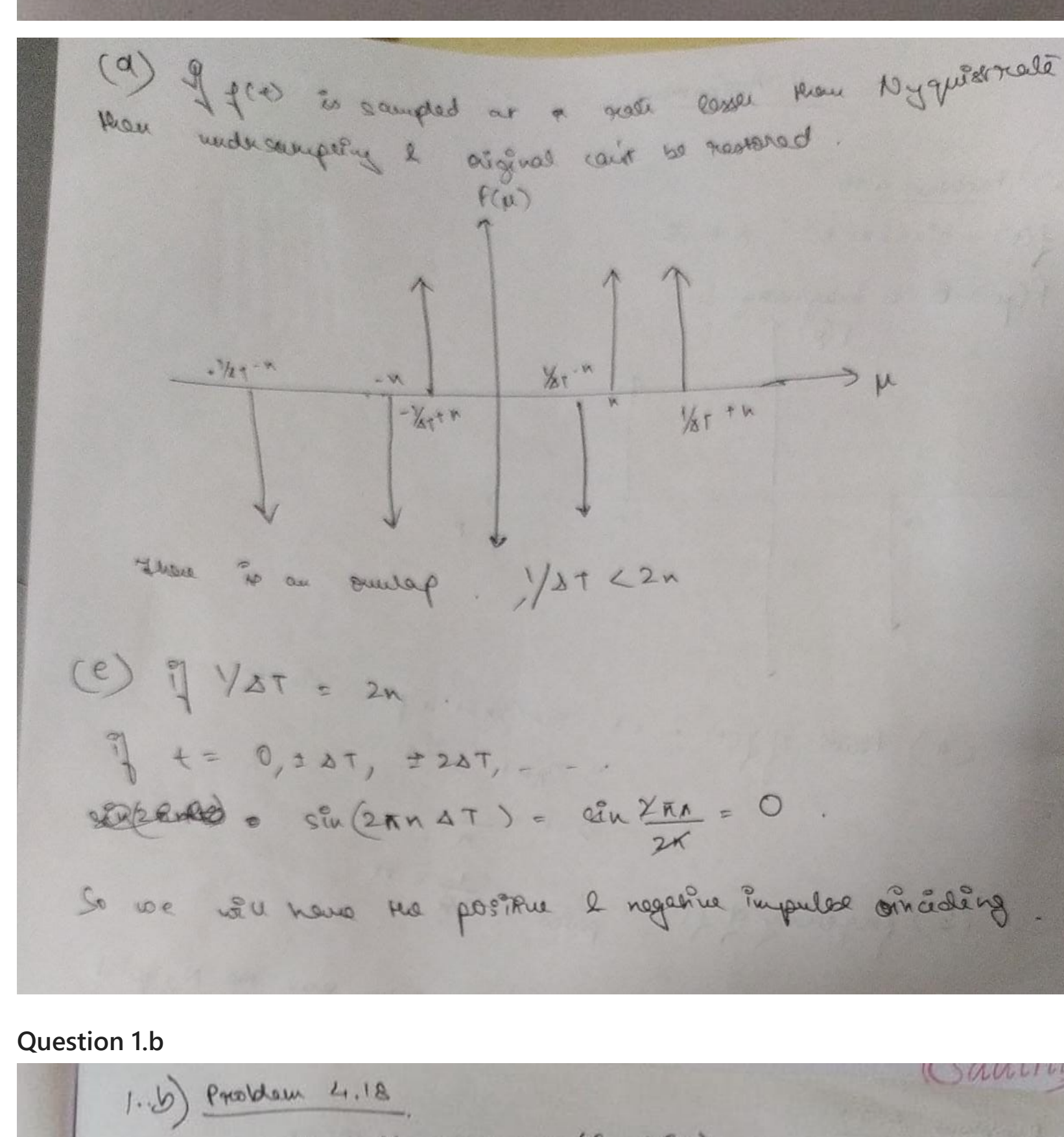
(b) Frequency of $f(t) = \frac{1}{\text{period}} = \frac{1}{1/n} = n$

(c) If $f(t)$ is sampled at a rate higher than the Nyquist rate then we are doing oversampling.

Nyquist rate $= 2n$

We will have superposition of ... the above diagram

where $1/\Delta T$ is the rate at which $f(t)$ is sampled & $\frac{1}{\Delta T} > 2n$


(d) If $f(t)$ is sampled at a rate lower than Nyquist rate then undersampling & original signal can't be restored.

There is an overlap $1/\Delta T < 2n$

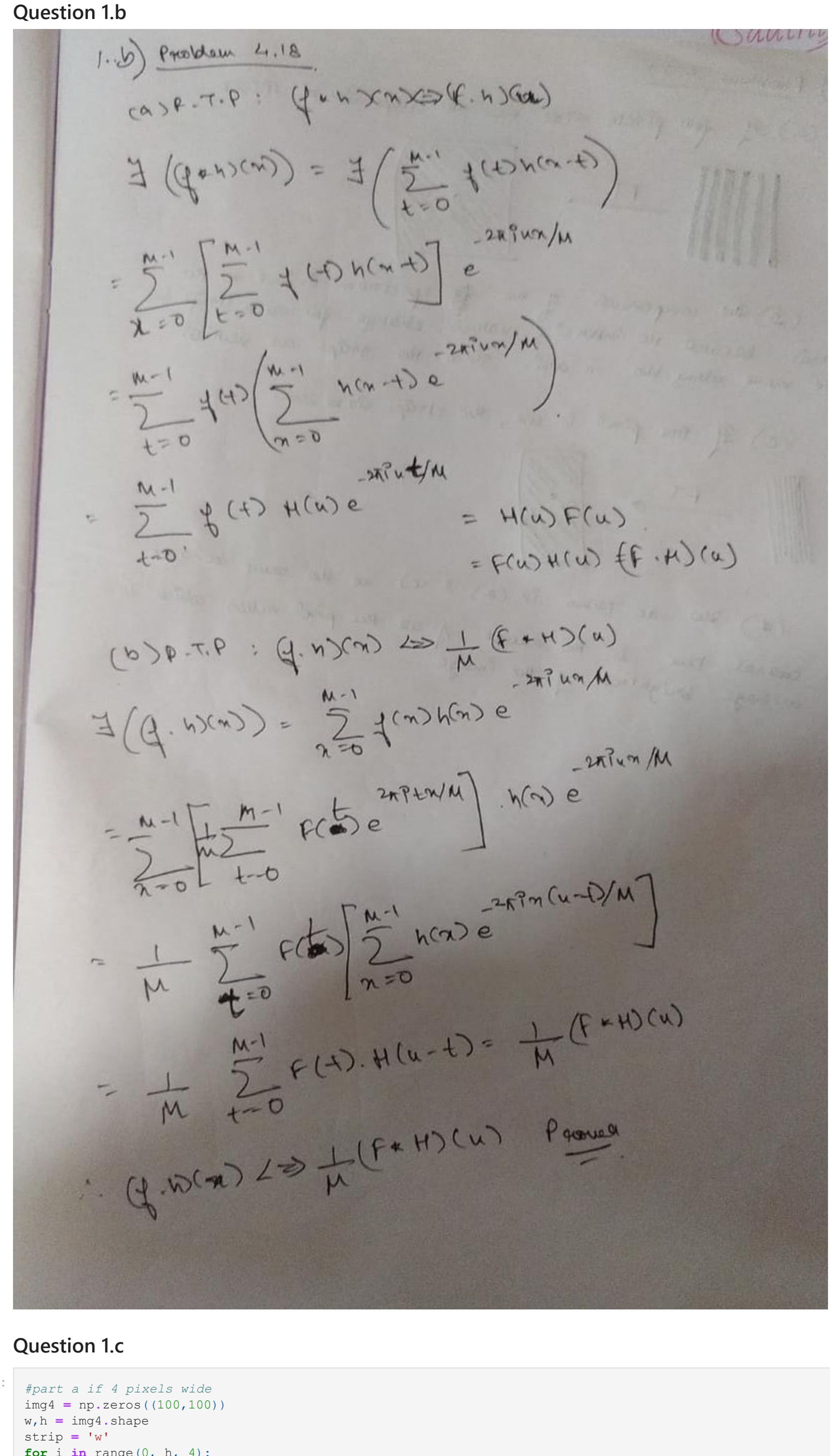
(e) If $1/\Delta T = 2n$

If $t = 0, \pm\Delta T, \pm 2\Delta T, \ldots$

$f(t) = \sin(2\pi n \Delta T) = \sin\frac{2\pi n k}{2n} = 0$

So we will have the positive & negative impulses overlapping.

Question 1.b


1.(b) Problem 4.18

(a) P.T.P: $(f \star h)(x) \Leftrightarrow (F \cdot H)(u)$

$\mathfrak{F}((f\star h)(x)) = \mathfrak{F}\left(\sum_{t=0}^{M-1} f(t)h(x-t)\right)$

$= \sum_{x=0}^{M-1}\left[\sum_{t=0}^{M-1} f(t)h(x-t)\right]e^{-j2\pi ux/M}$

$= \sum_{t=0}^{M-1} f(t)\left(\sum_{x=0}^{M-1} h(x-t)e^{-j2\pi ux/M}\right)$

$= \sum_{t=0}^{M-1} f(t)H(u)e^{-j2\pi ut/M} = H(u)F(u) = F(u)H(u)\ (F\cdot H)(u)$

(b) P.T.P: $(f\cdot h)(x) \Leftrightarrow \frac{1}{M}(F \star H)(u)$

$\mathfrak{F}((f\cdot h)(x)) = \sum_{x=0}^{M-1} f(x)h(x)e^{-j2\pi ux/M}$

$= \sum_{x=0}^{M-1}\left[\frac{1}{M}\sum_{t=0}^{M-1} F(t)e^{j2\pi tx/M}\right]h(x)e^{-j2\pi ux/M}$

$= \frac{1}{M}\sum_{t=0}^{M-1} F(t)\left[\sum_{x=0}^{M-1} h(x)e^{-j2\pi x(u-t)/M}\right]$

$= \frac{1}{M}\sum_{t=0}^{M-1} F(t)H(u-t) = \frac{1}{M}(F\star H)(u)$

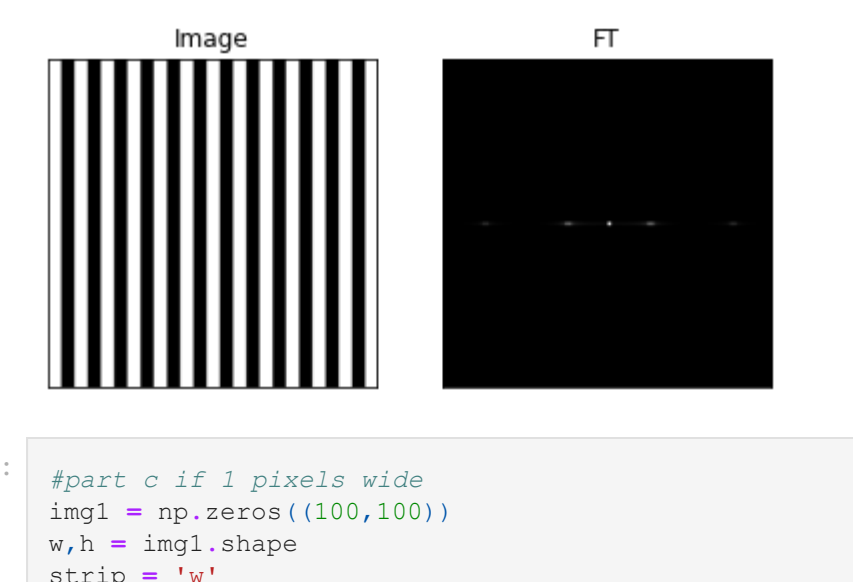
$\therefore (f\cdot h)(x) \Leftrightarrow \frac{1}{M}(F\star H)(u)$ Proved

Question 1.c

In [13]:
```
#part a if 4 pixels wide
img4 = np.zeros((100,100))
w,h = img4.shape
strip = 'w'
for i in range(0, h, 4):
    if strip == 'b':
        img4[:,i:(i+4)]=0
        strip='w'
    else:
        img4[:,i:(i+4)]=255
        strip='b'

img_f4 = np.abs(np.fft.fftshift(np.fft.fft2(img4)))
```

In [15]:
```
plt.figure(figsize=(10, 10))
plt.subplot(131),plt.imshow(img4, cmap = 'gray')
plt.title('Image'), plt.xticks([]), plt.yticks([])
plt.subplot(132),plt.imshow(img_f4, cmap = 'gray')
plt.title('FT'), plt.xticks([]), plt.yticks([])
```

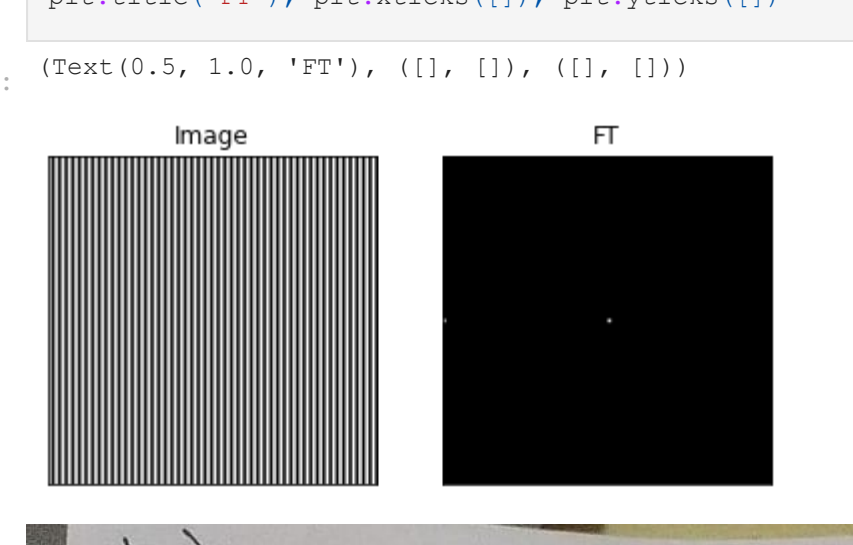
Out[15]: (Text(0.5, 1.0, 'FT'), ([], []), ([], []))

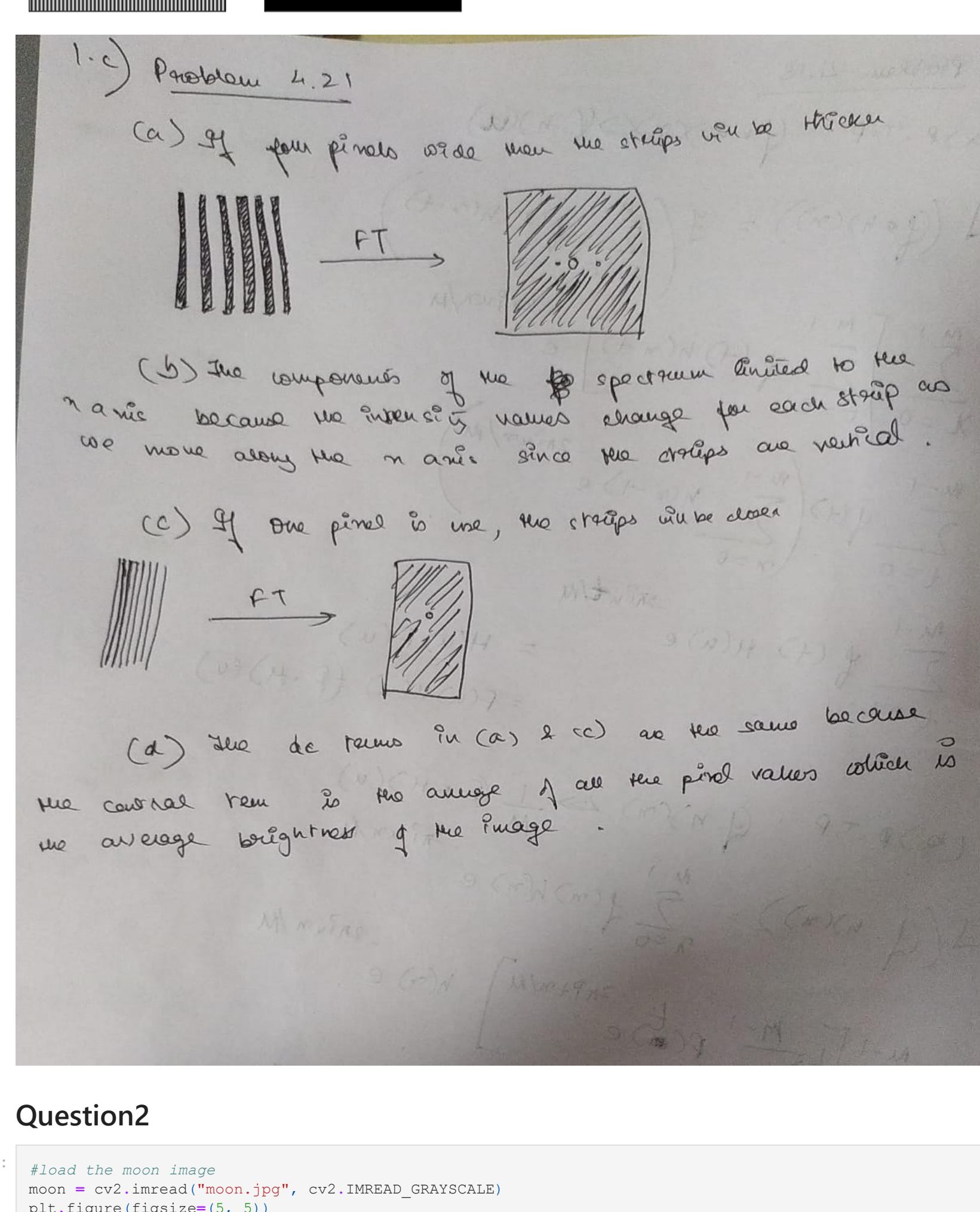


In [16]:
```
#part c if 1 pixels wide
img1 = np.zeros((100,100))
w,h = img1.shape
strip = 'w'
for i in range(0, h, 1):
    if strip == 'b':
        img1[:,i:(i+1)]=0
        strip='w'
    else:
        img1[:,i:(i+1)]=255
        strip='b'

img_f1 = np.abs(np.fft.fftshift(np.fft.fft2(img1)))
```

In [17]:
```
plt.figure(figsize=(10, 10))
plt.subplot(131),plt.imshow(img1, cmap = 'gray')
plt.title('Image'), plt.xticks([]), plt.yticks([])
plt.subplot(132),plt.imshow(img_f1, cmap = 'gray')
plt.title('FT'), plt.xticks([]), plt.yticks([])
```

Out[17]: (Text(0.5, 1.0, 'FT'), ([], []), ([], []))




1.c) Problem 4.21

(a) If four pixels wide then the strips will be thicker.

FT

(b) The components of the spectrum limited to the x axis because the intensity values change for each strip as we move along the x axis since the strips are vertical.

(c) If one pixel is used, the strips will be closer.

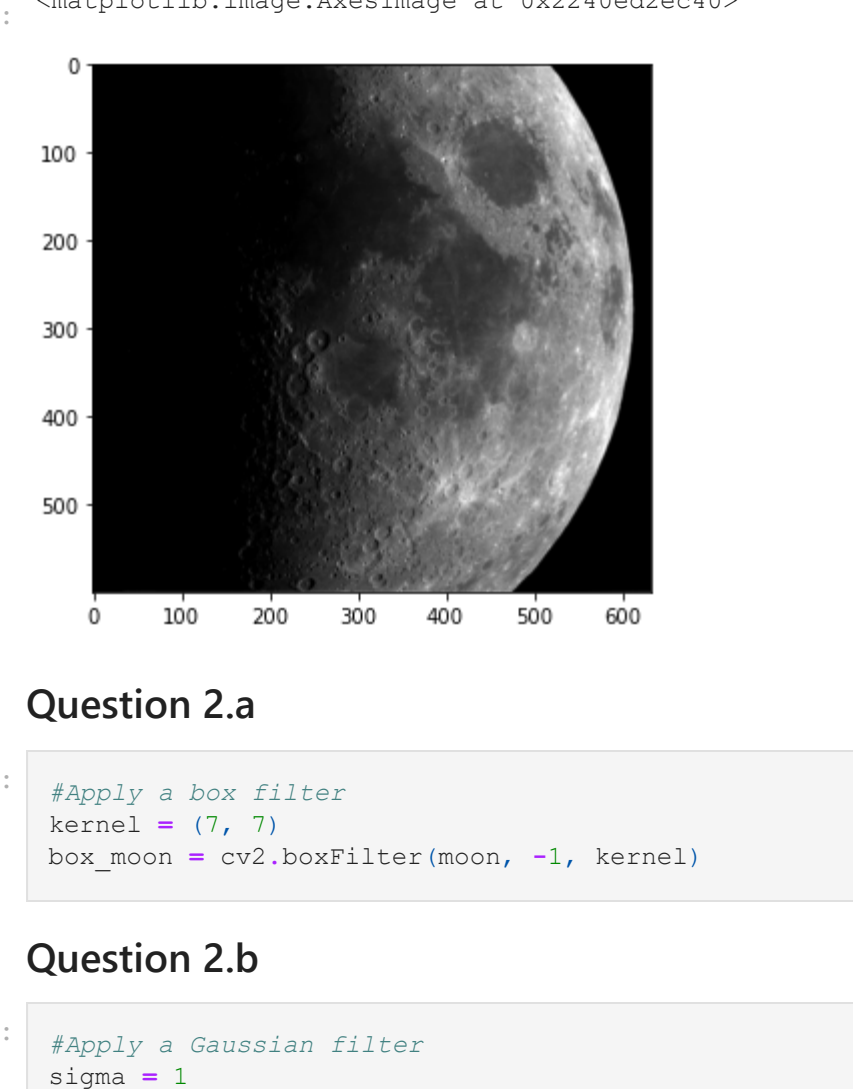
FT

(d) The dc terms in (a) & (c) are the same because the central term is the average of all the pixel values which is the average brightness of the image.

Question2

In [2]:
```
#load the moon image
moon = cv2.imread("moon.jpg", cv2.IMREAD_GRAYSCALE)
plt.figure(figsize=(5, 5))
plt.imshow(moon, "gray")
```

Out[2]: <matplotlib.image.AxesImage at 0x2240ed2ec40>



Question 2.a

In [3]:
```
#Apply a box filter
kernel = (7, 7)
box_moon = cv2.boxFilter(moon, -1, kernel)
```

Question 2.b

In [4]:
```
#Apply a Gaussian filter
sigma = 1
gaussian_moon = cv2.GaussianBlur(moon, kernel, sigma)
```

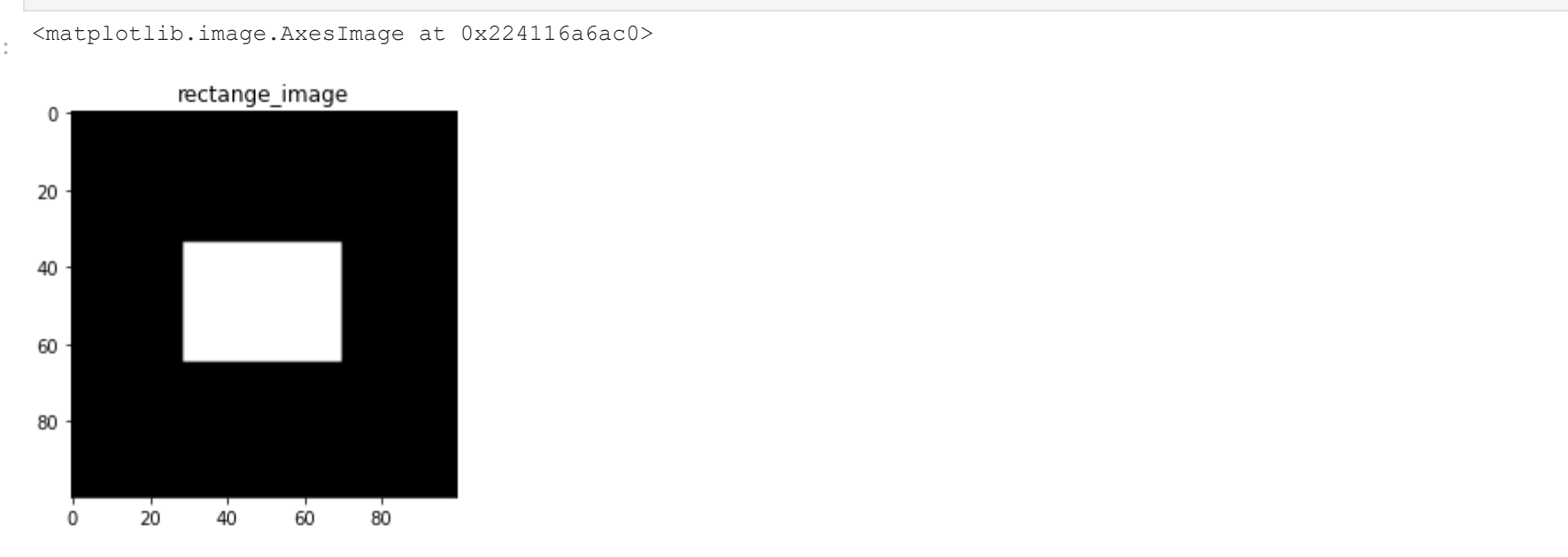
Question 2.c

In [5]:
```
# Apply a Laplacian filter
laplacian_moon = cv2.Laplacian(moon, cv2.CV_64F)

# Convert the output back to uint8
laplacian_moon = cv2.convertScaleAbs(laplacian_moon) + moon
```

Question 2.d

In [6]:
```
#Apply the sobel filter
sobel_x = cv2.Sobel(moon, cv2.CV_64F, 1, 0, ksize=5)
sobel_y = cv2.Sobel(moon, cv2.CV_64F, 0, 1, ksize=5)

# Take the absolute value of the filtered images and convert them to uint8
abs_sobel_x = np.uint8(np.absolute(sobel_x))
abs_sobel_y = np.uint8(np.absolute(sobel_y))

# Combine the x and y gradients using bitwise OR
sobel_moon = cv2.bitwise_or(abs_sobel_x, abs_sobel_y)
```

In [7]:
```
#plotting the different filtered images of moon
filtered_moon = {'moon':moon,'box_moon':box_moon, "gaussian_moon":gaussian_moon,"laplacian_moon":laplacian_moon
fig = plt.figure(figsize=(20, 10))
n = len(filtered_moon)
rows = 1
columns = 5
labels =list(filtered_moon.keys())
for j in range(n):
    fig.add_subplot(rows, columns, j+1)
    plt.imshow(filtered_moon[labels[j]],"gray")
    plt.title(labels[j])
    plt.axis("off")
```



Box filter: Has caused blurring of the image Gaussian filter: Smoothened the image Laplacian filter: Enhanced the edges and contrast Sobel filter: Resulted in a black and white image

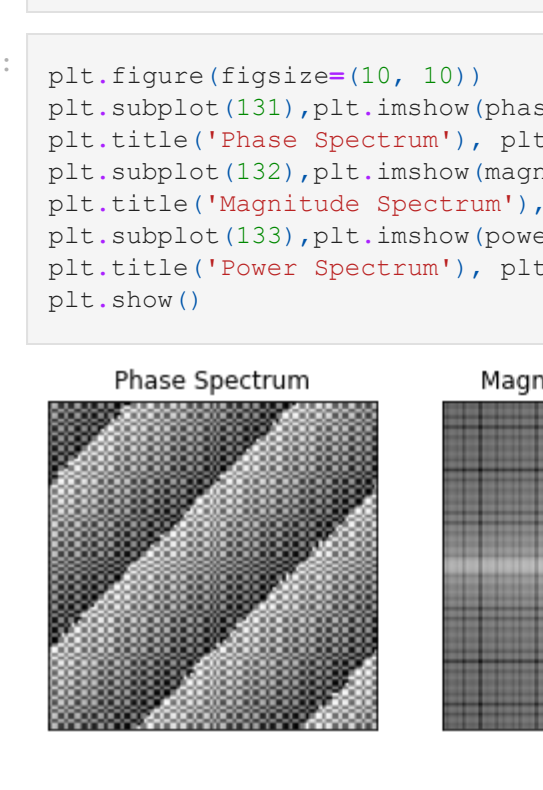
Question 3

In [8]:
```
#Displaying a white rectangle of any size x*y
def white_rect(x, y):
    img=np.zeros((100,100), dtype=np.uint8)
    img[(50-y//2):(50+y//2), (50-x//2):(50+x//2)] = 255
    return img
```

In [9]:
```
#White rectangle of size 40*30
img = white_rect(40,30)
```

In [10]:
```
plt.title("rectangle_image")
plt.imshow(img, "gray")
```

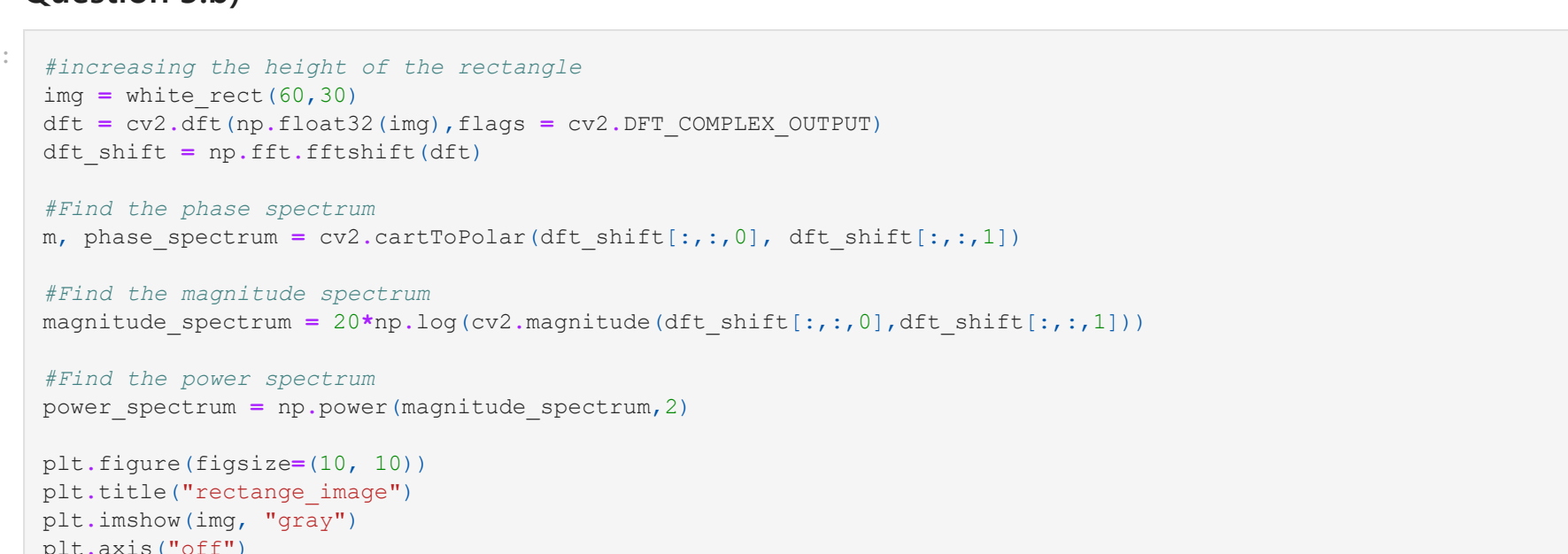
Out[10]: <matplotlib.image.AxesImage at 0x224116a6ac0>



Question 3.a

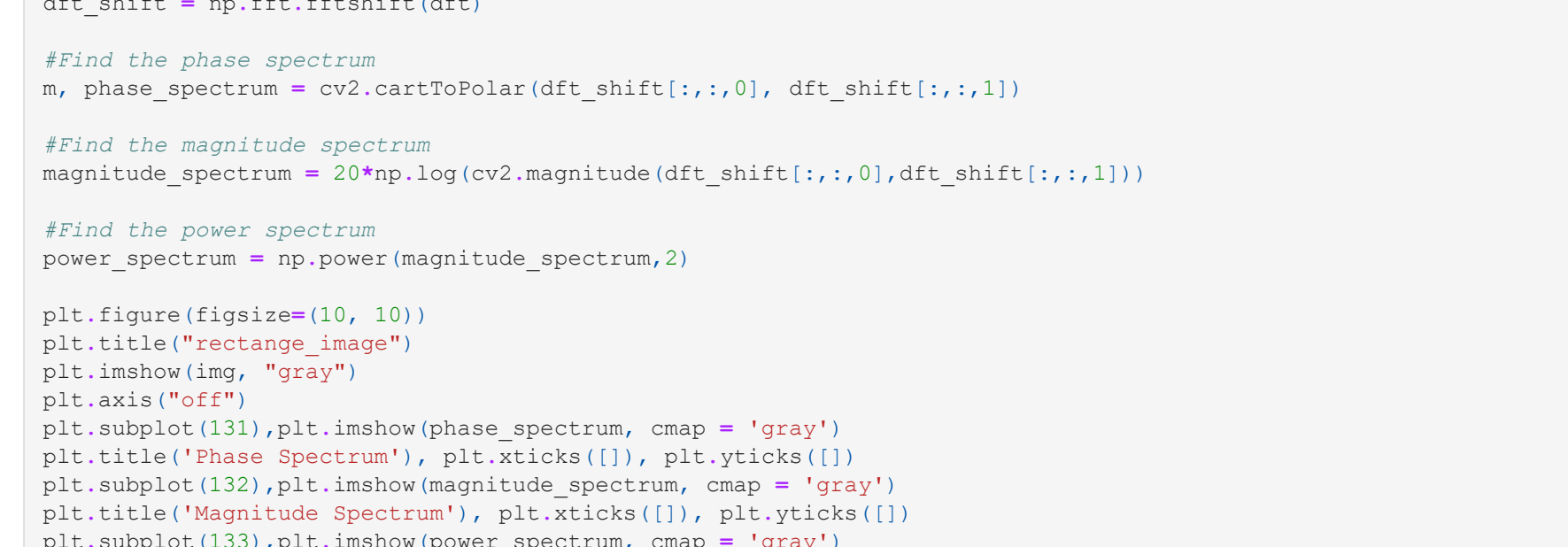
In [11]:
```
dft = cv2.dft(np.float32(img),flags = cv2.DFT_COMPLEX_OUTPUT)
dft_shift = np.fft.fftshift(dft)

#Find the phase spectrum
m, phase_spectrum = cv2.cartToPolar(dft_shift[:,:,0], dft_shift[:,:,1])

#Find the magnitude spectrum
magnitude_spectrum = 20*np.log(cv2.magnitude(dft_shift[:,:,0],dft_shift[:,:,1]))

#Find the power spectrum
power_spectrum = np.power(magnitude_spectrum,2)
```

In [12]:
```
plt.figure(figsize=(10, 10))
plt.subplot(131),plt.imshow(phase_spectrum, cmap = 'gray')
plt.title('Phase Spectrum'), plt.xticks([]), plt.yticks([])
plt.subplot(132),plt.imshow(magnitude_spectrum, cmap = 'gray')
plt.title('Magnitude Spectrum'), plt.xticks([]), plt.yticks([])
plt.subplot(133),plt.imshow(power_spectrum, cmap = 'gray')
plt.title('Power Spectrum'), plt.xticks([]), plt.yticks([])
plt.show()
```

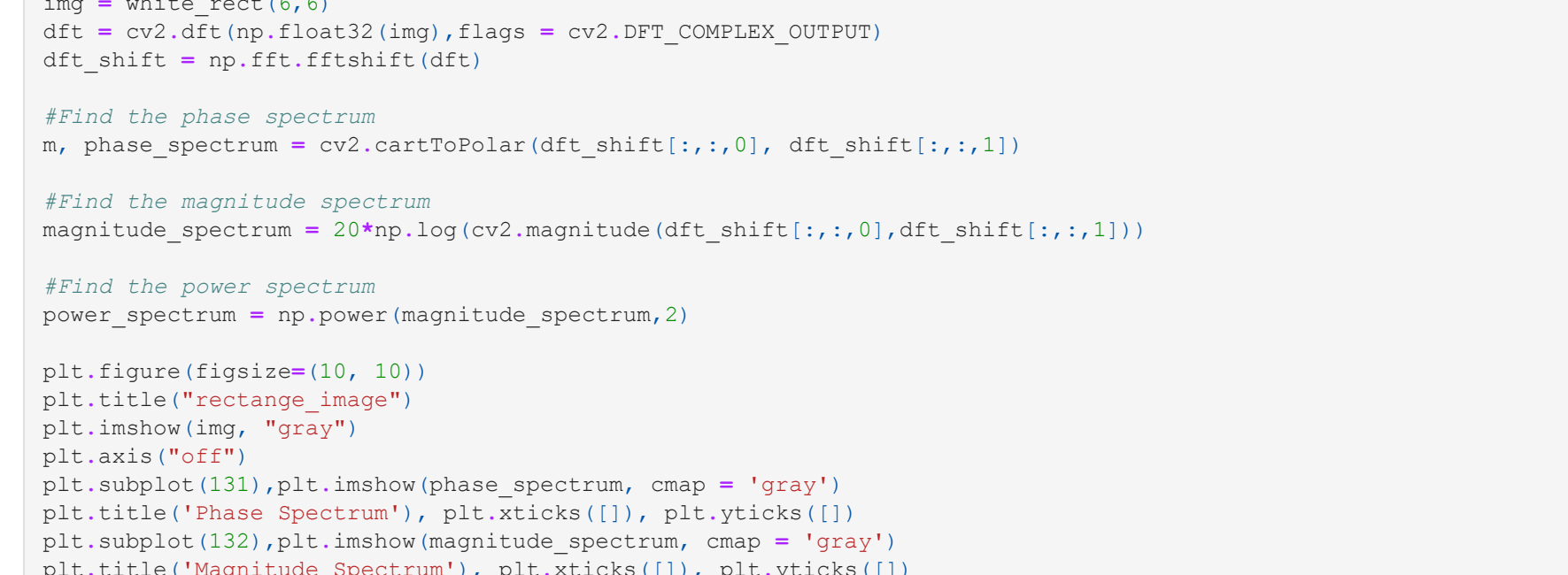


Question 3.b)

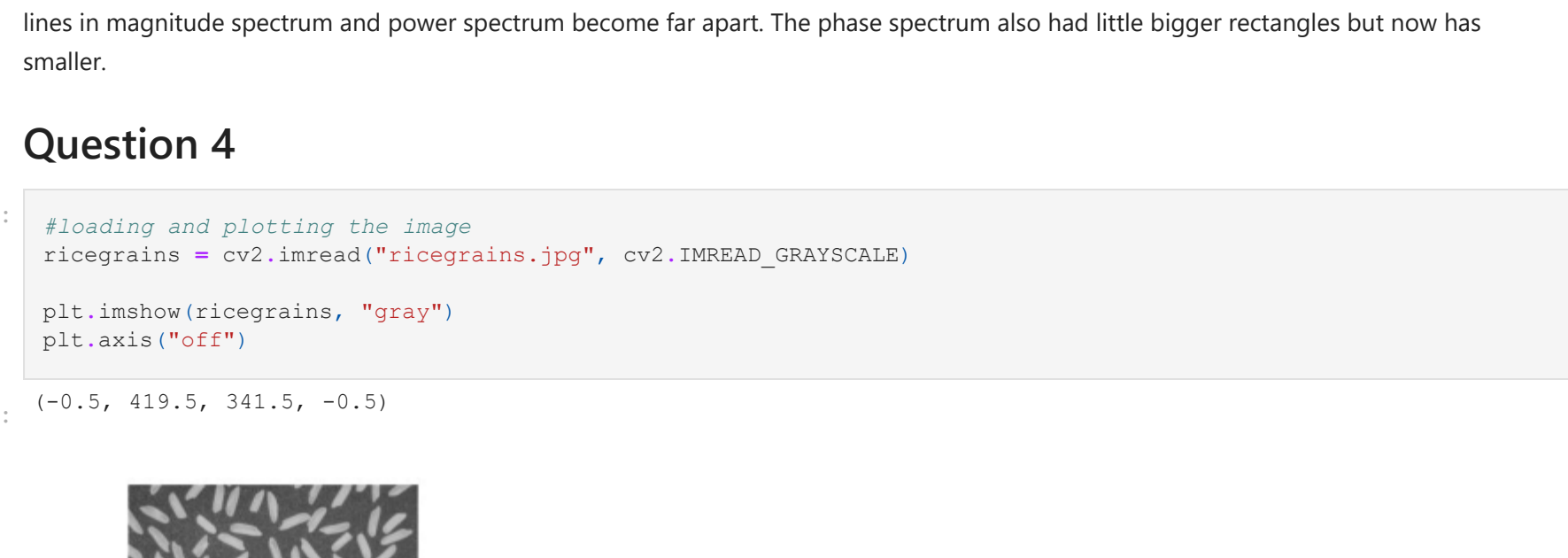
In [13]:
```
#increasing the height of the rectangle
img = white_rect(60,30)
dft = cv2.dft(np.float32(img),flags = cv2.DFT_COMPLEX_OUTPUT)
dft_shift = np.fft.fftshift(dft)

#Find the phase spectrum
m, phase_spectrum = cv2.cartToPolar(dft_shift[:,:,0], dft_shift[:,:,1])

#Find the magnitude spectrum
magnitude_spectrum = 20*np.log(cv2.magnitude(dft_shift[:,:,0],dft_shift[:,:,1]))

#Find the power spectrum
power_spectrum = np.power(magnitude_spectrum,2)

plt.figure(figsize=(10, 10))
plt.title("rectange_image")
plt.imshow(img, "gray")
plt.axis("off")
plt.subplot(131),plt.imshow(phase_spectrum, cmap = 'gray')
plt.title('Phase Spectrum'), plt.xticks([]), plt.yticks([])
plt.subplot(132),plt.imshow(magnitude_spectrum, cmap = 'gray')
plt.title('Magnitude Spectrum'), plt.xticks([]), plt.yticks([])
plt.subplot(133),plt.imshow(power_spectrum, cmap = 'gray')
plt.title('Power Spectrum'), plt.xticks([]), plt.yticks([])
plt.show()
```



In [14]:
```
#increasing both the height and width
img = white_rect(60,60)
dft = cv2.dft(np.float32(img),flags = cv2.DFT_COMPLEX_OUTPUT)
dft_shift = np.fft.fftshift(dft)

#Find the phase spectrum
m, phase_spectrum = cv2.cartToPolar(dft_shift[:,:,0], dft_shift[:,:,1])

#Find the magnitude spectrum
magnitude_spectrum = 20*np.log(cv2.magnitude(dft_shift[:,:,0],dft_shift[:,:,1]))

#Find the power spectrum
power_spectrum = np.power(magnitude_spectrum,2)

plt.figure(figsize=(10, 10))
plt.title("rectange_image")
plt.imshow(img, "gray")
plt.axis("off")
plt.subplot(131),plt.imshow(phase_spectrum, cmap = 'gray')
plt.title('Phase Spectrum'), plt.xticks([]), plt.yticks([])
plt.subplot(132),plt.imshow(magnitude_spectrum, cmap = 'gray')
plt.title('Magnitude Spectrum'), plt.xticks([]), plt.yticks([])
plt.subplot(133),plt.imshow(power_spectrum, cmap = 'gray')
plt.title('Power Spectrum'), plt.xticks([]), plt.yticks([])
plt.show()
```



In [15]:
```
#Decreasng the height and width
img = white_rect(6,6)
dft = cv2.dft(np.float32(img),flags = cv2.DFT_COMPLEX_OUTPUT)
dft_shift = np.fft.fftshift(dft)

#Find the phase spectrum
m, phase_spectrum = cv2.cartToPolar(dft_shift[:,:,0], dft_shift[:,:,1])

#Find the magnitude spectrum
magnitude_spectrum = 20*np.log(cv2.magnitude(dft_shift[:,:,0],dft_shift[:,:,1]))

#Find the power spectrum
power_spectrum = np.power(magnitude_spectrum,2)

plt.figure(figsize=(10, 10))
plt.title("rectange_image")
plt.imshow(img, "gray")
plt.axis("off")
plt.subplot(131),plt.imshow(phase_spectrum, cmap = 'gray')
plt.title('Phase Spectrum'), plt.xticks([]), plt.yticks([])
plt.subplot(132),plt.imshow(magnitude_spectrum, cmap = 'gray')
plt.title('Magnitude Spectrum'), plt.xticks([]), plt.yticks([])
plt.subplot(133),plt.imshow(power_spectrum, cmap = 'gray')
plt.title('Power Spectrum'), plt.xticks([]), plt.yticks([])
plt.show()
```

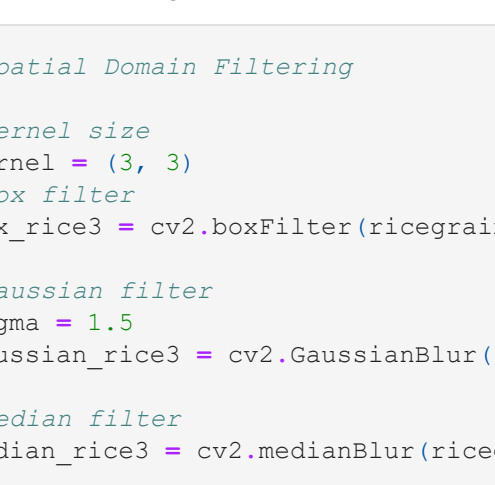


As we increase the rectangle size we can see that te black line become more prominent. As we decrease the rectangle size, we can see the lines in magnitude spectrum and power spectrum become far apart. The phase spectrum also had little bigger rectangles but now has smaller.

Question 4

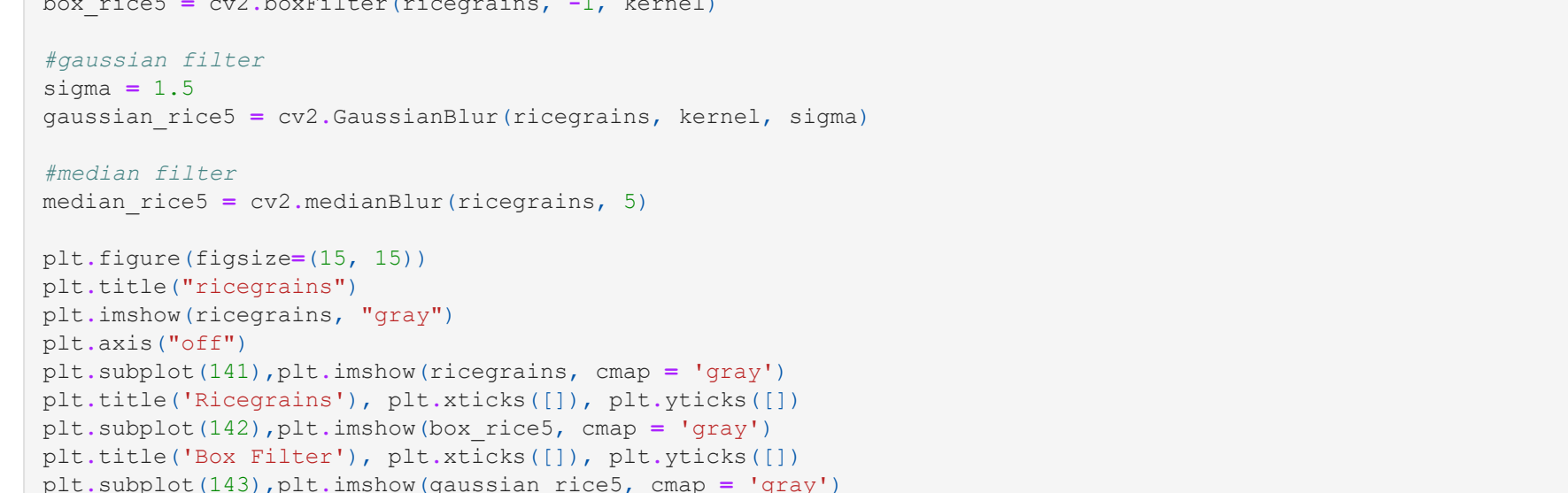
In [16]:
```
#loading and plotting the image
ricegrains = cv2.imread("ricegrains.jpg", cv2.IMREAD_GRAYSCALE)

plt.imshow(ricegrains, "gray")
plt.axis("off")
```

Out[16]: (-0.5, 419.5, 341.5, -0.5)

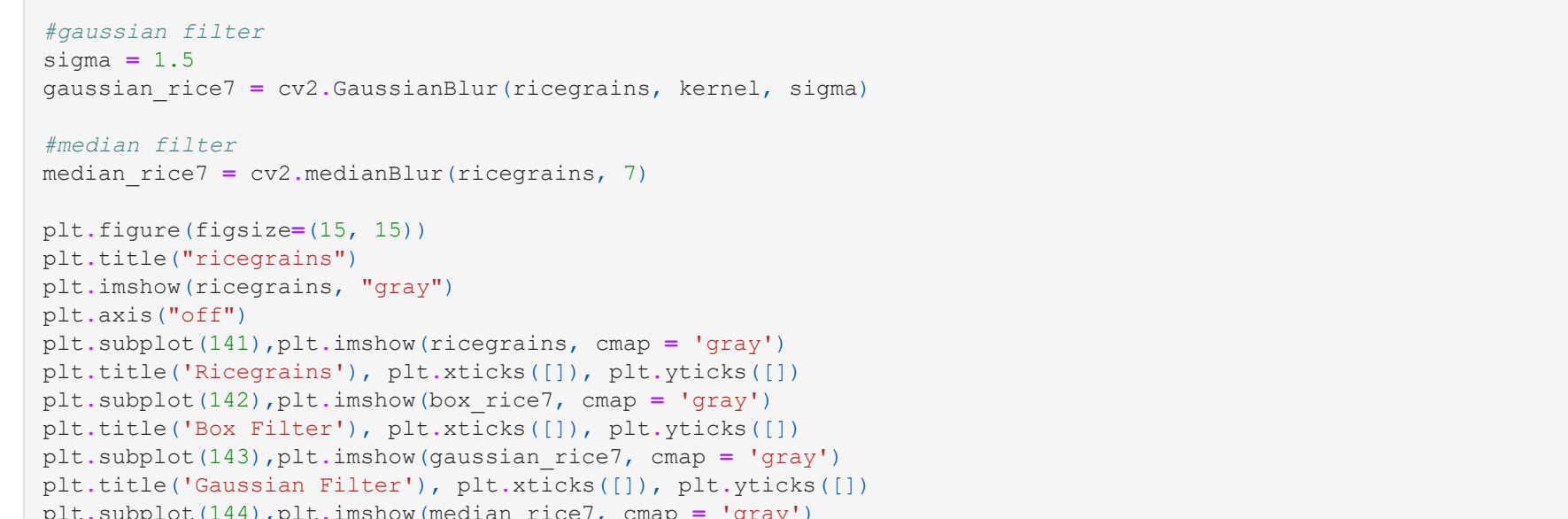


Question 4.a)

In [17]:
```
#Spatial Domain Filtering

#kernel size
kernel = (3, 3)
#box filter
box_rice3 = cv2.boxFilter(ricegrains, -1, kernel)

#gaussian filter
sigma = 1.5
gaussian_rice3 = cv2.GaussianBlur(ricegrains, kernel, sigma)

#median filter
median_rice3 = cv2.medianBlur(ricegrains, 3)

plt.figure(figsize=(15, 15))
plt.title("ricegrains")
plt.imshow(ricegrains, "gray")
plt.axis("off")
plt.subplot(141),plt.imshow(ricegrains, cmap = "gray")
plt.title('Ricegrains'), plt.xticks([]), plt.yticks([])
plt.subplot(142),plt.imshow(box_rice3, cmap = "gray")
plt.title('Box Filter'), plt.xticks([]), plt.yticks([])
plt.subplot(143),plt.imshow(gaussian_rice3, cmap = "gray")
plt.title('Gaussian Filter'), plt.xticks([]), plt.yticks([])
plt.subplot(144),plt.imshow(median_rice3, cmap = "gray")
plt.title('Median Filter'), plt.xticks([]), plt.yticks([])
plt.show()
```



In [18]:
```
#kernel size
kernel = (5, 5)

#box filter
box_rice5 = cv2.boxFilter(ricegrains, -1, kernel)

#gaussian filter
sigma = 1.5
gaussian_rice5 = cv2.GaussianBlur(ricegrains, kernel, sigma)

#median filter
median_rice5 = cv2.medianBlur(ricegrains, 5)

plt.figure(figsize=(15, 15))
plt.title("ricegrains")
plt.imshow(ricegrains, "gray")
plt.axis("off")
plt.subplot(141),plt.imshow(ricegrains, cmap = "gray")
plt.title('Ricegrains'), plt.xticks([]), plt.yticks([])
plt.subplot(142),plt.imshow(box_rice5, cmap = "gray")
plt.title('Box Filter'), plt.xticks([]), plt.yticks([])
plt.subplot(143),plt.imshow(gaussian_rice5, cmap = "gray")
plt.title('Gaussian Filter'), plt.xticks([]), plt.yticks([])
plt.subplot(144),plt.imshow(median_rice5, cmap = "gray")
plt.title('Median Filter'), plt.xticks([]), plt.yticks([])
plt.show()
```



In [19]:
```
#kernel size
kernel = (7, 7)

#box filter
box_rice7 = cv2.boxFilter(ricegrains, -1, kernel)

#gaussian filter
sigma = 1.5
gaussian_rice7 = cv2.GaussianBlur(ricegrains, kernel, sigma)

#median filter
median_rice7 = cv2.medianBlur(ricegrains, 7)

plt.figure(figsize=(15, 15))
plt.title("ricegrains")
plt.imshow(ricegrains, "gray")
plt.axis("off")
plt.subplot(141),plt.imshow(ricegrains, cmap = "gray")
plt.title('Ricegrains'), plt.xticks([]), plt.yticks([])
plt.subplot(142),plt.imshow(box_rice7, cmap = "gray")
plt.title('Box Filter'), plt.xticks([]), plt.yticks([])
plt.subplot(143),plt.imshow(gaussian_rice7, cmap = "gray")
plt.title('Gaussian Filter'), plt.xticks([]), plt.yticks([])
plt.subplot(144),plt.imshow(median_rice7, cmap = "gray")
plt.title('Median Filter'), plt.xticks([]), plt.yticks([])
plt.show()
```

As we increase the kernel size, the filters produces a more blurry image.

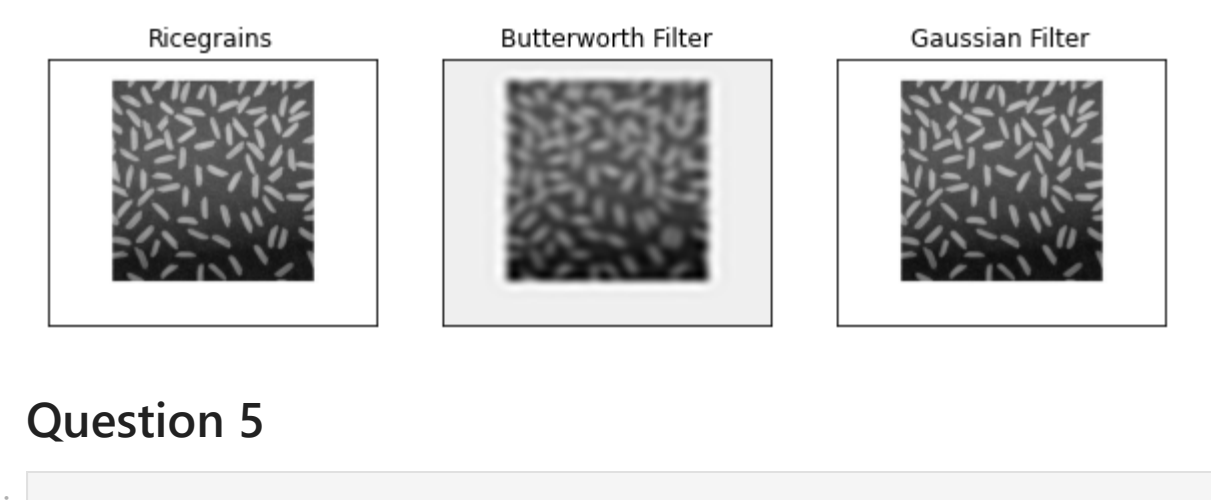
Question 4.b)

In [20]:
```
#Butterworth lowpass frequency filter

#Fourier transform of image
fshift_bw = np.fft.fftshift(np.fft.fft2(ricegrains))
h,w = ricegrains.shape
img_low_bw = np.zeros((h,w)) #defining the output
K_low_bw = np.zeros((h,w)) #kernel

for i in range(h):
    for j in range(w):
        dist = np.sqrt((i-h/2)**2 + (j-w/2)**2)
        K_low_bw[i,j] = 1/(1+(dist/(20)**(2*4)))

#applying the filter to the transformed image
fshift_low_bw = fshift_bw * K_low_bw
img_low_bw = np.abs(np.fft.ifft2(fshift_low_bw))
```

In [21]:
```
#Gaussian lowpass frequency filter

#Fourier transform of image
fshift_g = np.fft.fftshift(np.fft.fft2(ricegrains))
h,w = ricegrains.shape
img_low_g = np.zeros((h,w)) #defining the output
K_low_g = np.zeros((h,w)) #kernel

for i in range(h):
    for j in range(w):
        dist = np.sqrt((i-h/2)**2 + (j-w/2)**2)
        K_low_g[i,j] = np.exp(-(dist**2)/(2*(20**2)))

#applying the filter to the transformed image
fshift_low_g = fshift_g * K_low_g
img_low_g = np.abs(np.fft.ifft2(fshift_low_g))
```

In [22]:
```
plt.figure(figsize=(10, 10))
plt.subplot(131),plt.imshow(ricegrains, cmap = "gray")
plt.title('Ricegrains'), plt.xticks([]), plt.yticks([])
plt.subplot(132),plt.imshow(img_low_bw, cmap = "gray")
plt.title('Butterworth Filter'), plt.xticks([]), plt.yticks([])
plt.subplot(133),plt.imshow(img_low_g.astype(np.uint8), cmap = "gray")
plt.title('Gaussian Filter'), plt.xticks([]), plt.yticks([])
```

Out[22]: (Text(0.5, 1.0, 'Gaussian Filter'), ([], []), ([], []))



Question 5

In [23]:
```
#loading and plotting the image
tiger = cv2.imread("tigerbw.jpg", cv2.IMREAD_GRAYSCALE)

plt.imshow(tiger, "gray")
plt.axis("off")
```

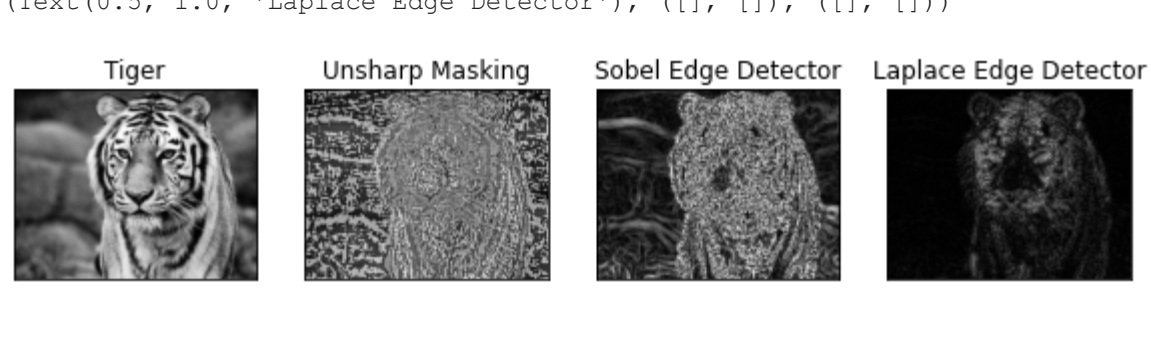
Out[23]: (-0.5, 235.5, 184.5, -0.5)

Question 5.a

In [24]:
```
#unsharp masking
K=1.5
kernel = cv2.getGaussianKernel(5, 5)
blurred = cv2.filter2D(tiger, -1, kernel * kernel.T)
mask = tiger-blurred
tiger_um = tiger + K*mask

#Sobel Edge Detector
sobel_x = cv2.Sobel(tiger, cv2.CV_64F, 1, 0, ksize=3)
sobel_y = cv2.Sobel(tiger, cv2.CV_64F, 0, 1, ksize=3)
sobel = np.sqrt(sobel_x**2 + sobel_y**2)
sobel = np.uint8(sobel)

#Laplace Edge detector
blur = cv2.GaussianBlur(tiger, (3, 3), 0)
laplace = cv2.Laplacian(blur, cv2.CV_64F)
laplace = np.uint8(np.absolute(laplace))
```

In [25]:
```
plt.figure(figsize=(10, 10))
plt.subplot(141),plt.imshow(tiger, cmap = 'gray')
plt.title('Tiger'), plt.xticks([]), plt.yticks([])
plt.subplot(142),plt.imshow(tiger_um, cmap = 'gray')
plt.title('Unsharp Masking'), plt.xticks([]), plt.yticks([])
plt.subplot(143),plt.imshow(sobel, cmap = 'gray')
plt.title('Sobel Edge Detector'), plt.xticks([]), plt.yticks([])
plt.subplot(144),plt.imshow(laplace, cmap = 'gray')
plt.title('Laplace Edge Detector'), plt.xticks([]), plt.yticks([])
```

Out[25]: (Text(0.5, 1.0, 'Laplace Edge Detector'), ([], []), ([], []))



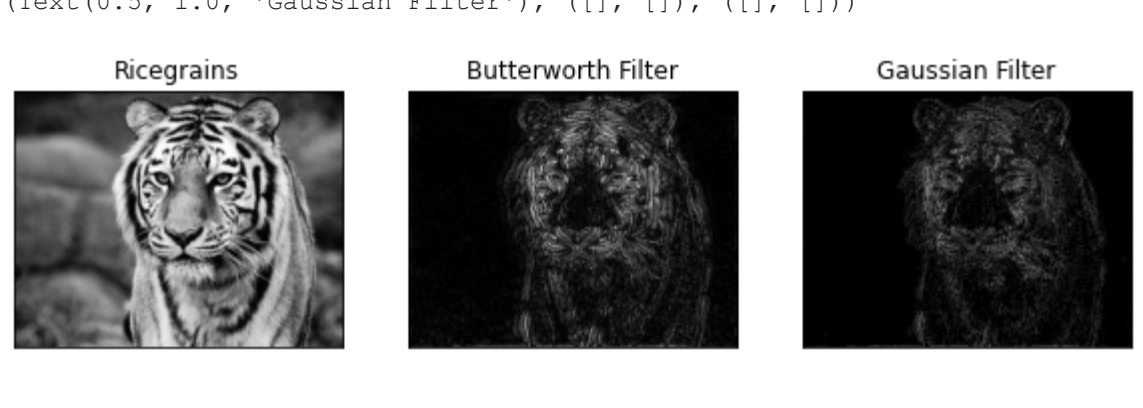
Question 5.b

In [26]:
```
#Butterworth highpass frequency filter

#fourier transform of the image
fshift_bw = np.fft.fftshift(np.fft.fft2(tiger))
h,w = tiger.shape
img_high_bw = np.zeros((h,w))
K_high_bw = np.zeros((h,w))

for i in range(h):
    for j in range(w):
      dist = np.sqrt((i-h//2)**2 + (j-w//2)**2)
      K_high_bw[i,j] = 1/(1+((dist/20)**(2*4)))
img_high_bw = np.abs(np.fft.ifft2(fshift_bw*(1-K_high_bw)))
```

In [27]:
```
#Gaussian highpass frequency filter

#fourier transform of the image
fshift_g = np.fft.fftshift(np.fft.fft2(tiger))
h,w = tiger.shape
img_high_g = np.zeros((h,w))
K_high_g = np.zeros((h,w))

for i in range(h):
    for j in range(w):
      dist = np.sqrt((i-h//2)**2 + (j-w//2)**2)
      K_high_g[i,j] = np.exp(-(dist/(2*(20**2))))

img_high_g = np.abs(np.fft.ifft2(fshift_g*(1-K_high_g)))
```

In [28]:
```
plt.figure(figsize=(10, 10))
plt.subplot(131),plt.imshow(tiger, cmap = 'gray')
plt.title('Ricegrains'), plt.xticks([]), plt.yticks([])
plt.subplot(132),plt.imshow(img_high_bw, cmap = 'gray')
plt.title('Butterworth Filter'), plt.xticks([]), plt.yticks([])
plt.subplot(133),plt.imshow(img_high_g.astype(np.uint8), cmap = 'gray')
plt.title('Gaussian Filter'), plt.xticks([]), plt.yticks([])
```

Out[28]: (Text(0.5, 1.0, 'Gaussian Filter'), ([], []), ([], []))



Question 6

In [29]:
```
#oading and plotting the image
cameraman = cv2.imread("cameraman.jpg", cv2.IMREAD_GRAYSCALE)

plt.imshow(cameraman, "gray")
plt.axis("off")
```

Out[29]: (-0.5, 224.5, 224.5, -0.5)

Question 6.a

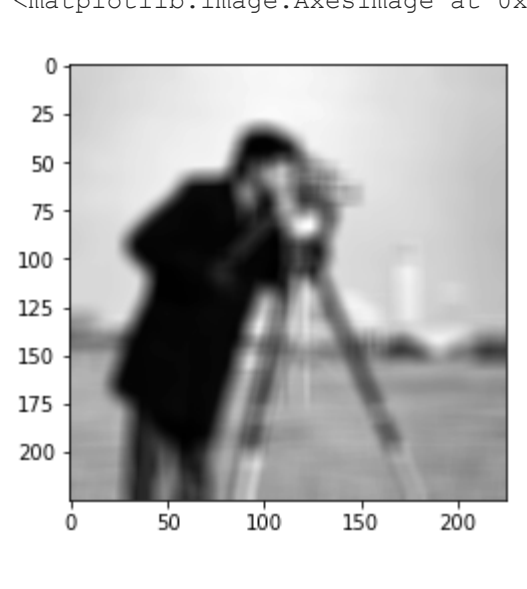
In [38]:
```
#average filter of size 11*11
kernel_size = (11, 11)
kernel = np.ones(kernel_size, np.float32) / (kernel_size[0] * kernel_size[1])
from scipy.signal import convolve2d
```

In [41]:
```
import time
start = time.time()
#applying the average filter
avg = convolve2d(cameraman, kernel, mode="same")
end = time.time()
print(end-start)
```

0.0398249626159668

In [32]:
```
#plotting the result
plt.imshow(avg,'gray')
```

Out[32]: <matplotlib.image.AxesImage at 0x224117ac250>



Question 6.b

In [33]:
```
#Padding the image and the kernel
P,Q = 2*cameraman.shape[0],2*cameraman.shape[1]
cameraman_pad = np.zeros((P, Q), dtype=np.uint8)
cameraman_pad[:cameraman.shape[0],:cameraman.shape[1]] = cameraman
kernel_pad = np.zeros((P, Q), dtype=np.float32)
kernel_pad[:kernel.shape[0],:kernel.shape[1]] = kernel

#Fourier Transform of the image and the kernel
cameraman_f = np.fft.fftshift(np.fft.fft2(cameraman_pad))
kernel_f = np.fft.fftshift(np.fft.fft2(kernel_pad))
```

In [40]:
```
import time
start = time.time()
#Multiplying them
multi = np.multiply(cameraman_f, kernel_f)
end = time.time()
print(end-start)
```

0.008026123046875

In [35]:
```
#Inverse fourier transform
cameraman_if = np.abs(np.fft.ifft2(multi))

#useful content of transformed image
cameraman_cut = cameraman_if.astype(np.uint8)[:cameraman.shape[0],:cameraman.shape[1]]
```

In [36]:
```
#plotting the transformed image
plt.imshow(cameraman_cut, "gray")
plt.axis("off")
```

Out[36]: (-0.5, 224.5, 224.5, -0.5)

Question 6.c

Time required for part a is: 0.0398 s and for part b is: 0.008026 s \ Multiplication in the frequency domain is 5 times faster than Convolution in the spatial domain.

In [ ]: