DIP Assignment-

Group No 7 Members

->PALETI KRISHNASAI- CED18I039

->KATTE PRAHLAD GOWTHAM - COE18B029

->HRISHIKESH RAJESH MENON - COE18B024

Questions

1. Download Lena color image convert it to grayscale image and add salt and pepper noise with noise quantity 0.1,0.2 upto 1 and generate 10 noisy images.

   a. Do average filtering (by correlating with average filter) of varying sizes for each image. Filter size can be 3x3, 5x5, 7x7. (In 3x3 filter all the values are 1/9, in 5x5 filter all the values are 1/25 and in 7x7 filter all the values are 1/49)

   b. Similarly, repeat the question 1.a by replacing the average filter by median filter.

1. a. Consider the image the attached named as hdraw.png and crop each of the characters from the image and consider that as the sub-image. Find the location of the sub-image in the original by using correlation.

b. Download Lena color image convert it to grayscale image and crop the left eye of Lena as sub-image and do the cross-correlation (Normalized correlation) to find the location of the left eye in the original image.

1. Write a function to implement FFT for 1D signal.
2. Implement DFT function for an image using the FFT for 1D signal using question 3.
3. Consider the images of lena and dog images attached. Find phase and magnitude of the dog and lena images using DFT function implemented in question 4.
4. Swap phase of the dog image and magnitude of the lena image and display the output.
5. Swap phase of the lena image and magnitude of the dog image ad display the output

Note: For all the above questions write user-defined functions and compare with predefined (in-built) function.

Question 1

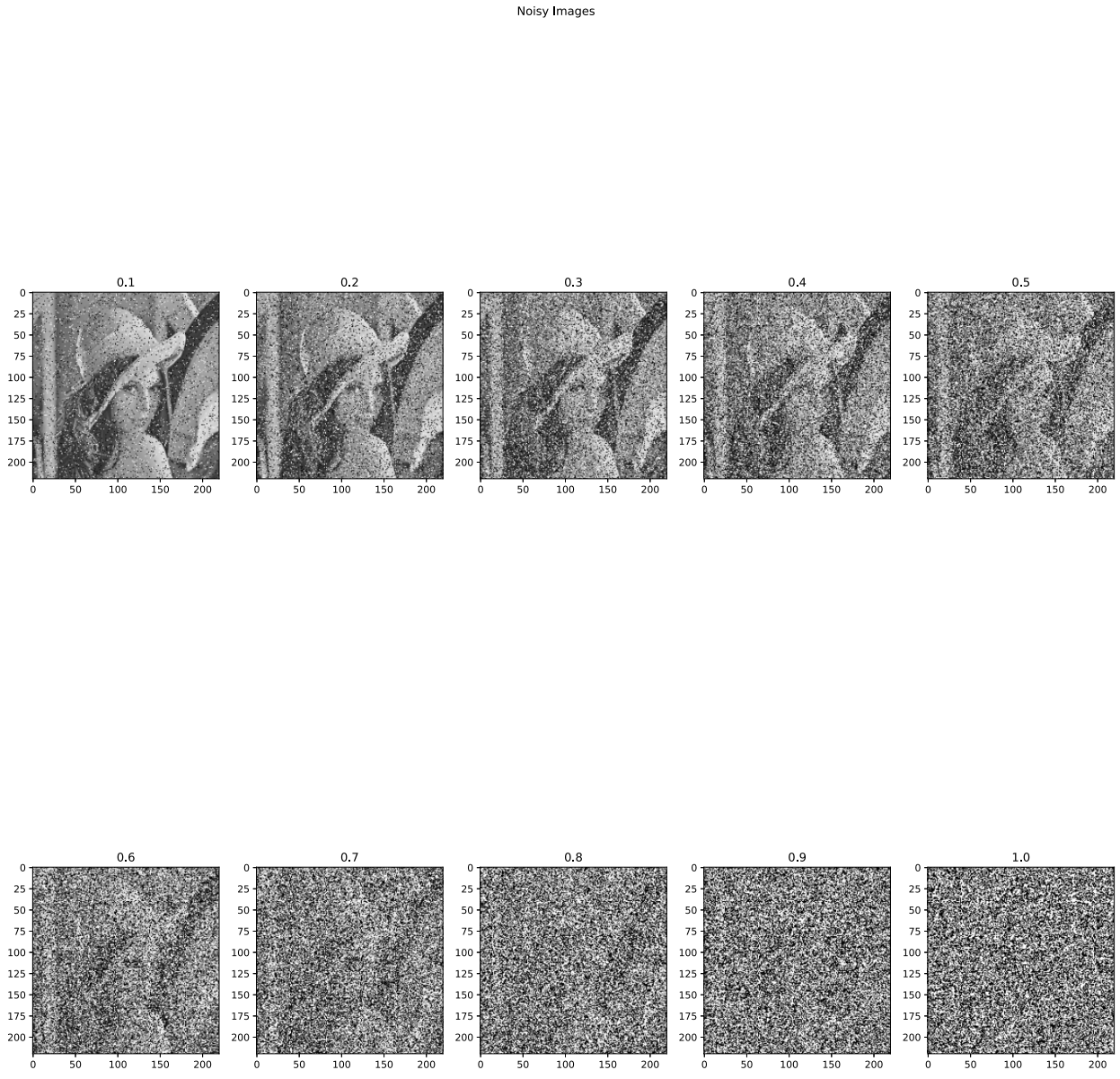
Download Lena color image convert it to grayscale image and add salt and pepper noise with noise quantity 0.1,0.2 upto 1 and generate 10 noisy images.

In [7]:

```
import cv2
from matplotlib import pyplot as plt
from skimage.util import random_noise
import numpy as np

img = cv2.imread("lena.png")
gray = cv2.imread("lena.png",0)

#Salt and Pepper
def sp(n):
    noise_img=random_noise(gray,mode='s&p',amount=n)
    noise_img=np.array(255*noise_img,dtype='uint8')
    cv2.imwrite('sp_42049.png',noise_img)

    return noise_img

a1=sp(0.1)
a2=sp(0.2)
a3=sp(0.3)
a4=sp(0.4)
a5=sp(0.5)
```

```
a6=sp(0.6)
a7=sp(0.7)
a8=sp(0.8)
a9=sp(0.9)
a10=sp(1)


fig,axs = plt.subplots(2,5,figsize=(20,20))

plt.suptitle("Noisy Images")

axs[0,0].set_title("0.1")
axs[0,0].imshow(cv2.cvtColor(a1, cv2.COLOR_BGR2RGB))
axs[0,1].set_title("0.2")
axs[0,1].imshow(cv2.cvtColor(a2, cv2.COLOR_BGR2RGB))
axs[0,2].set_title("0.3")
axs[0,2].imshow(cv2.cvtColor(a3, cv2.COLOR_BGR2RGB))
axs[0,3].set_title("0.4")
axs[0,3].imshow(cv2.cvtColor(a4, cv2.COLOR_BGR2RGB))
axs[0,4].set_title("0.5")
axs[0,4].imshow(cv2.cvtColor(a5, cv2.COLOR_BGR2RGB))
axs[1,0].set_title("0.6")
axs[1,0].imshow(cv2.cvtColor(a6, cv2.COLOR_BGR2RGB))
axs[1,1].set_title("0.7")
axs[1,1].imshow(cv2.cvtColor(a7, cv2.COLOR_BGR2RGB))
axs[1,2].set_title("0.8")
axs[1,2].imshow(cv2.cvtColor(a8, cv2.COLOR_BGR2RGB))
axs[1,3].set_title("0.9")
axs[1,3].imshow(cv2.cvtColor(a9, cv2.COLOR_BGR2RGB))
axs[1,4].set_title("1.0")
axs[1,4].imshow(cv2.cvtColor(a10, cv2.COLOR_BGR2RGB))
```

Out[7]: <matplotlib.image.AxesImage at 0x1d05cb10f10>



Do average filtering (by correlating with average filter) of varying sizes for each image. Filter size can be 3x3, 5x5, 7x7. (In 3x3 filter all the values are 1/9, in 5x5 filter all the values are 1/25 and in 7x7 filter all the values are 1/49)

Function

In [9]:

```
def Avg_Filter(imag,si):
    # print(imag.shape)
    f1=np.zeros([si,si])
    f1.fill(1/(si*si))
    # print("f1 is :\n ",f1)
    f1_new=np.zeros([imag.shape[0]+f1.shape[0]-1,imag.shape[1]+f1.shape[1]-1])#new matrix of size m+a-1 and n+b-1
    # print(f1_new.shape)
    n1=np.insert(imag, 0, 0, axis = 1)
    n1=np.insert(n1, 0, 0, axis = 0)

    if si==3:
        n1=np.insert(n1,a1.shape[0]+1, 0, axis = 1)
        n1=np.insert(n1,a1.shape[0]+1,0,axis=0)
        dec=1
    elif si==5 :
        n1=np.insert(n1, 1, 0, axis = 1)
        n1=np.insert(n1,1,0,axis=0)
        n1=np.insert(n1,a1.shape[0]+2,0,axis=1)
        n1=np.insert(n1,a1.shape[0]+3,0,axis=1)
        n1=np.insert(n1,a1.shape[0]+2, 0, axis = 0)
        n1=np.insert(n1,a1.shape[0]+3,0,axis=0)
        dec=2
    elif si==7 :
        n1=np.insert(n1,1,0, axis = 1)
        n1=np.insert(n1,1,0,axis=0)
        n1=np.insert(n1,0,0,axis=1)
        n1=np.insert(n1,a1.shape[0]+1,0,axis=0)
        n1=np.insert(n1,a1.shape[0]+2,0,axis=1)
        n1=np.insert(n1,a1.shape[0]+3,0,axis=1)
        n1=np.insert(n1,a1.shape[0]+2,0, axis = 0)
        n1=np.insert(n1,a1.shape[0]+3,0,axis=0)
        n1=np.insert(n1,a1.shape[0]+4,0,axis=0)
        n1=np.insert(n1,a1.shape[0]+4,0,axis=1)
        dec=3


    # print("n1 is :\n",n1) # n1 contains the imag matrix with extra 2 rows and cols of 0's
    # print(n1.shape)
    # print(f1.shape)
    b=int((f1.shape[0]-1)/2)
    a=int((f1.shape[1]-1)/2)
    # print(imag)

    for x in range(0,imag.shape[0]+f1.shape[0]-dec):
        for y in range(0,imag.shape[1]+f1.shape[1]-dec):
    #         print("inside y")
    #         f1_new[x,y]=0
            for j in range(-1*b,b):
                for i in range(-1*a,a):

                    f1_new[x,y]+=np.multiply(n1[x+i,y+j],f1[i,j])
    #                 print(f1_new[x,y])

    return f1_new
```

3x3

In [10]:

```
#3x3 Average Filtering

fig,axs = plt.subplots(4,5,figsize=(20,20))

plt.suptitle("3x3 Average Filtering",fontsize=30)

axs[0,0].set_title("0.1 User Defined")
axs[0,0].imshow(Avg_Filter(a1,3),"gray")
axs[0,1].set_title("0.2 User Defined")
axs[0,1].imshow(Avg_Filter(a2,3),"gray")
axs[0,2].set_title("0.3 User Defined")
axs[0,2].imshow(Avg_Filter(a3,3),"gray")
axs[0,3].set_title("0.4 User Defined")
axs[0,3].imshow(Avg_Filter(a4,3),"gray")
axs[0,4].set_title("0.5 User Defined")
axs[0,4].imshow(Avg_Filter(a5,3),"gray")
axs[1,0].set_title("0.6 User Defined")
axs[1,0].imshow(Avg_Filter(a6,3),"gray")
axs[1,1].set_title("0.7 User Defined")
axs[1,1].imshow(Avg_Filter(a7,3),"gray")
axs[1,2].set_title("0.8 User Defined")
axs[1,2].imshow(Avg_Filter(a8,3),"gray")
axs[1,3].set_title("0.9 User Defined")
axs[1,3].imshow(Avg_Filter(a9,3),"gray")
axs[1,4].set_title("1.0 User Defined")
axs[1,4].imshow(Avg_Filter(a10,3),"gray")

#Inbuilt
axs[2,0].set_title("0.1 In-Built")
axs[2,0].imshow(cv2.blur(a1,(3,3)),"gray")
axs[2,1].set_title("0.2 In-Built")
axs[2,1].imshow(cv2.blur(a2,(3,3)),"gray")
axs[2,2].set_title("0.3 In-Built")
axs[2,2].imshow(cv2.blur(a3,(3,3)),"gray")
axs[2,3].set_title("0.4 In-Built")
axs[2,3].imshow(cv2.blur(a4,(3,3)),"gray")
axs[2,4].set_title("0.5 In-Built")
axs[2,4].imshow(cv2.blur(a5,(3,3)),"gray")
axs[3,0].set_title("0.6 In-Built")
axs[3,0].imshow(cv2.blur(a6,(3,3)),"gray")
axs[3,1].set_title("0.7 In-Built")
axs[3,1].imshow(cv2.blur(a7,(3,3)),"gray")
axs[3,2].set_title("0.8 In-Built")
axs[3,2].imshow(cv2.blur(a8,(3,3)),"gray")
axs[3,3].set_title("0.9 In-Built")
axs[3,3].imshow(cv2.blur(a9,(3,3)),"gray")
axs[3,4].set_title("1.0 In-Built")
axs[3,4].imshow(cv2.blur(a10,(3,3)),"gray")

plt.setp(plt.gcf().get_axes(), xticks=[], yticks=[])
```

Out[10]: []

3x3 Average Filtering

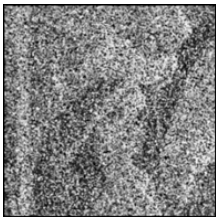
0.1 User Defined

0.2 User Defined

0.3 User Defined

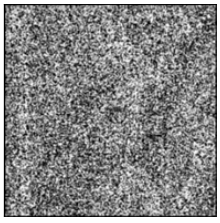
0.4 User Defined

0.5 User Defined

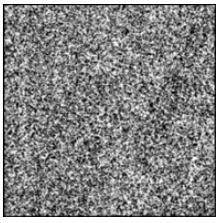


0.6 User Defined

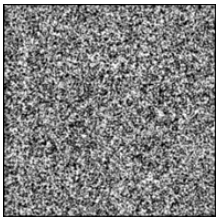
0.7 User Defined



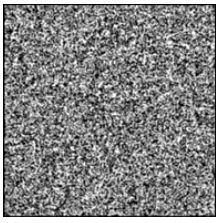
0.8 User Defined



0.9 User Defined



1.0 User Defined

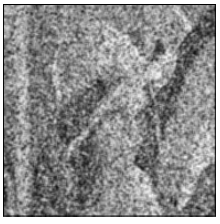


0.1 In-Built

0.2 In-Built

0.3 In-Built

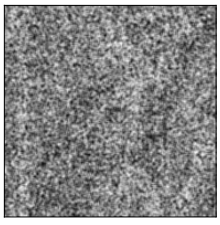
0.4 In-Built



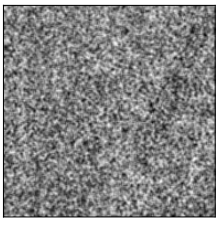
0.5 In-Built

0.6 In-Built

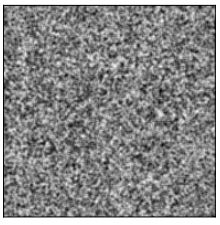
0.7 In-Built



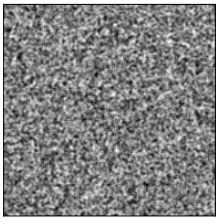
0.8 In-Built



0.9 In-Built



1.0 In-Built



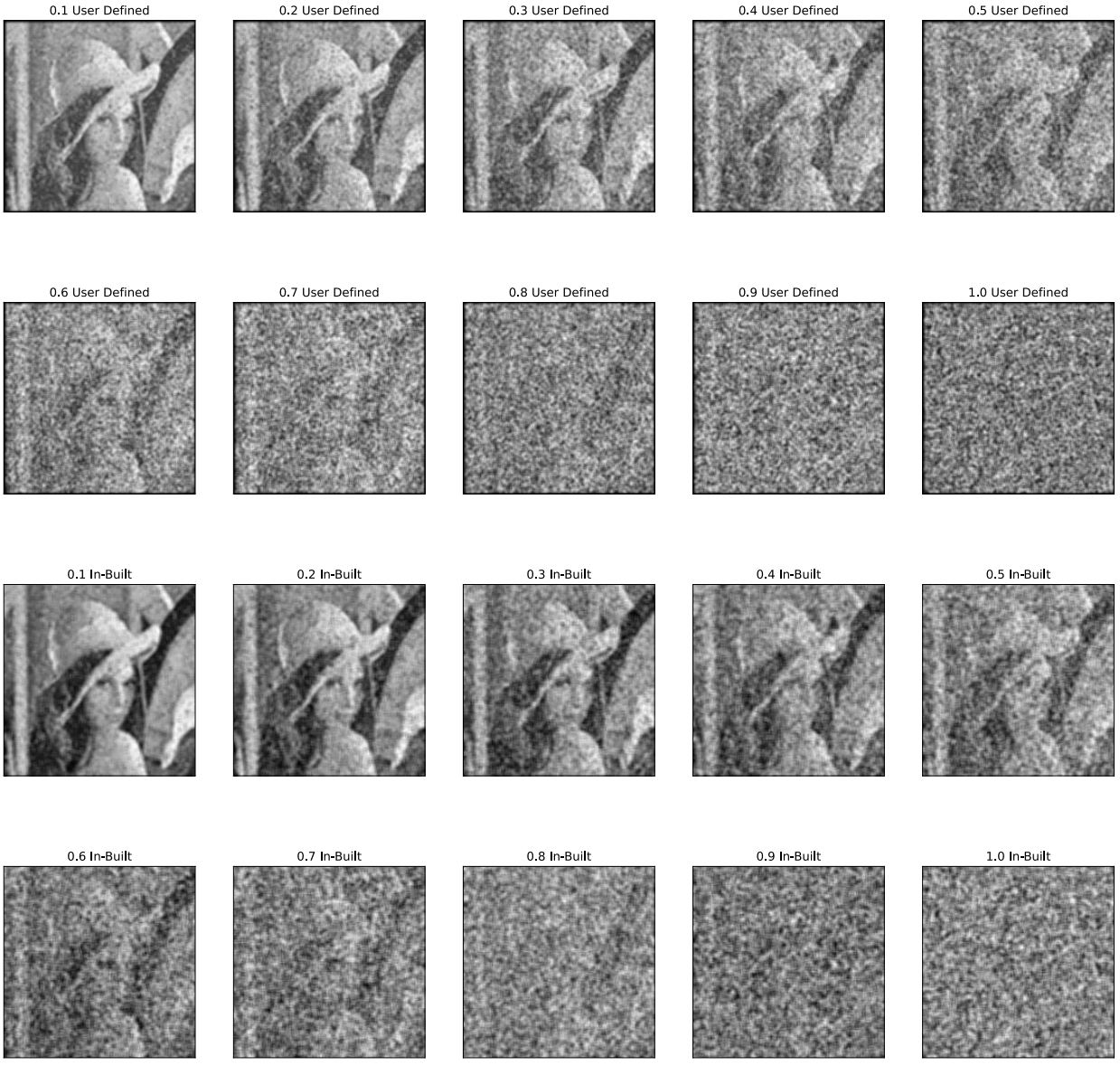
5x5

In [11]:

```
# 5x5 Filtering


fig,axs = plt.subplots(4,5,figsize=(20,20))

plt.suptitle("5x5 Average Filtering",fontsize=30)

axs[0,0].set_title("0.1 User Defined")
axs[0,0].imshow(Avg_Filter(a1,5),"gray")
axs[0,1].set_title("0.2 User Defined")
axs[0,1].imshow(Avg_Filter(a2,5),"gray")
axs[0,2].set_title("0.3 User Defined")
axs[0,2].imshow(Avg_Filter(a3,5),"gray")
axs[0,3].set_title("0.4 User Defined")
axs[0,3].imshow(Avg_Filter(a4,5),"gray")
axs[0,4].set_title("0.5 User Defined")
axs[0,4].imshow(Avg_Filter(a5,5),"gray")
axs[1,0].set_title("0.6 User Defined")
axs[1,0].imshow(Avg_Filter(a6,5),"gray")
axs[1,1].set_title("0.7 User Defined")
axs[1,1].imshow(Avg_Filter(a7,5),"gray")
axs[1,2].set_title("0.8 User Defined")
axs[1,2].imshow(Avg_Filter(a8,5),"gray")
axs[1,3].set_title("0.9 User Defined")
axs[1,3].imshow(Avg_Filter(a9,5),"gray")
axs[1,4].set_title("1.0 User Defined")
axs[1,4].imshow(Avg_Filter(a10,5),"gray")

#Inbuilt
axs[2,0].set_title("0.1 In-Built")
axs[2,0].imshow(cv2.blur(a1,(5,5)),"gray")
axs[2,1].set_title("0.2 In-Built")
axs[2,1].imshow(cv2.blur(a2,(5,5)),"gray")
axs[2,2].set_title("0.3 In-Built")
axs[2,2].imshow(cv2.blur(a3,(5,5)),"gray")
axs[2,3].set_title("0.4 In-Built")
axs[2,3].imshow(cv2.blur(a4,(5,5)),"gray")
axs[2,4].set_title("0.5 In-Built")
axs[2,4].imshow(cv2.blur(a5,(5,5)),"gray")
axs[3,0].set_title("0.6 In-Built")
axs[3,0].imshow(cv2.blur(a6,(5,5)),"gray")
axs[3,1].set_title("0.7 In-Built")
axs[3,1].imshow(cv2.blur(a7,(5,5)),"gray")
axs[3,2].set_title("0.8 In-Built")
axs[3,2].imshow(cv2.blur(a8,(5,5)),"gray")
axs[3,3].set_title("0.9 In-Built")
axs[3,3].imshow(cv2.blur(a9,(5,5)),"gray")
axs[3,4].set_title("1.0 In-Built")
axs[3,4].imshow(cv2.blur(a10,(5,5)),"gray")

plt.setp(plt.gcf().get_axes(), xticks=[], yticks=[])
```

Out[11]: []

5x5 Average Filtering

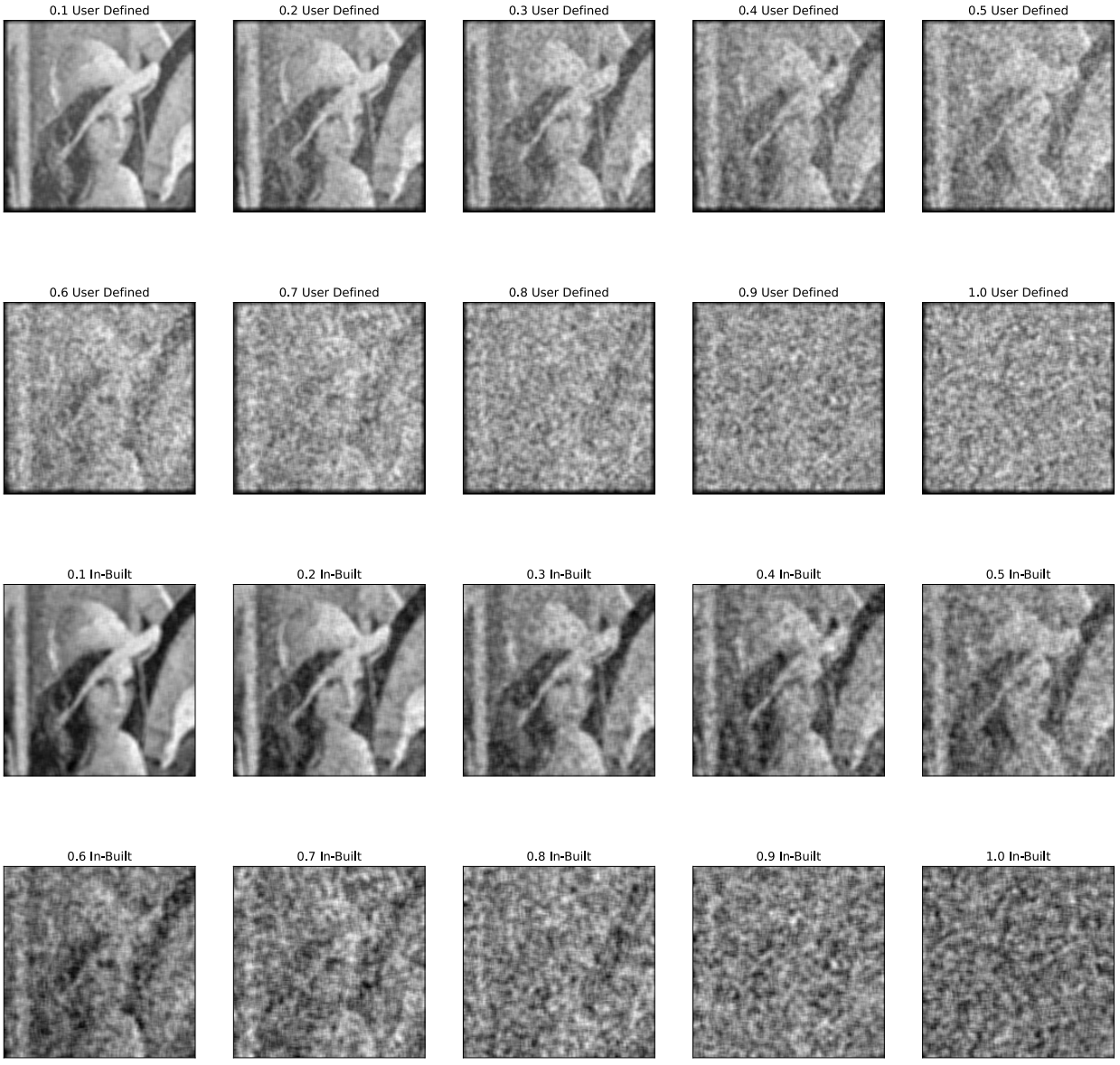


7x7

In [12]:
```
# 7x7 Filtering


fig,axs = plt.subplots(4,5,figsize=(20,20))

plt.suptitle("7x7 Average Filtering",fontsize=30)

axs[0,0].set_title("0.1 User Defined")
axs[0,0].imshow(Avg_Filter(a1,7),"gray")
axs[0,1].set_title("0.2 User Defined")
axs[0,1].imshow(Avg_Filter(a2,7),"gray")
axs[0,2].set_title("0.3 User Defined")
axs[0,2].imshow(Avg_Filter(a3,7),"gray")
axs[0,3].set_title("0.4 User Defined")
axs[0,3].imshow(Avg_Filter(a4,7),"gray")
axs[0,4].set_title("0.5 User Defined")
axs[0,4].imshow(Avg_Filter(a5,7),"gray")
axs[1,0].set_title("0.6 User Defined")
axs[1,0].imshow(Avg_Filter(a6,7),"gray")
axs[1,1].set_title("0.7 User Defined")
axs[1,1].imshow(Avg_Filter(a7,7),"gray")
axs[1,2].set_title("0.8 User Defined")
axs[1,2].imshow(Avg_Filter(a8,7),"gray")
axs[1,3].set_title("0.9 User Defined")
axs[1,3].imshow(Avg_Filter(a9,7),"gray")
axs[1,4].set_title("1.0 User Defined")
axs[1,4].imshow(Avg_Filter(a10,7),"gray")

#Inbuilt
axs[2,0].set_title("0.1 In-Built")
axs[2,0].imshow(cv2.blur(a1,(7,7)),"gray")
axs[2,1].set_title("0.2 In-Built")
axs[2,1].imshow(cv2.blur(a2,(7,7)),"gray")
axs[2,2].set_title("0.3 In-Built")
axs[2,2].imshow(cv2.blur(a3,(7,7)),"gray")
axs[2,3].set_title("0.4 In-Built")
axs[2,3].imshow(cv2.blur(a4,(7,7)),"gray")
axs[2,4].set_title("0.5 In-Built")
axs[2,4].imshow(cv2.blur(a5,(7,7)),"gray")
axs[3,0].set_title("0.6 In-Built")
axs[3,0].imshow(cv2.blur(a6,(7,7)),"gray")
axs[3,1].set_title("0.7 In-Built")
axs[3,1].imshow(cv2.blur(a7,(7,7)),"gray")
axs[3,2].set_title("0.8 In-Built")
axs[3,2].imshow(cv2.blur(a8,(7,7)),"gray")
axs[3,3].set_title("0.9 In-Built")
axs[3,3].imshow(cv2.blur(a9,(7,7)),"gray")
axs[3,4].set_title("1.0 In-Built")
axs[3,4].imshow(cv2.blur(a10,(7,7)),"gray")

plt.setp(plt.gcf().get_axes(), xticks=[], yticks=[])
```

Out[12]: []

7x7 Average Filtering



b. Similarly, repeat the question 1.a by replacing the average filter by median filter.

Function

In [1]:
```
def median_filter(imag, si):
    temp = []
    indexer = si // 2
    img_final = []
    img_final = np.zeros((len(imag),len(imag[0])))
    for i in range(len(imag)):

        for j in range(len(imag[0])):

            for z in range(si):
                if i + z - indexer < 0 or i + z - indexer > len(imag) - 1:
                    for c in range(si):
                        temp.append(0)
                else:
                    if j + z - indexer < 0 or j + indexer > len(imag[0]) - 1:
                        temp.append(0)
                    else:
                        for k in range(si):
                            temp.append(imag[i + z - indexer][j + k - indexer])

            temp.sort()
            img_final[i][j] = temp[len(temp) // 2]
            temp = []
    return img_final
```

3x3

In [13]:
```
#3x3 Median Filtering

fig,axs = plt.subplots(4,5,figsize=(20,20))

plt.suptitle("3x3 Median Filtering",fontsize=30)

axs[0,0].set_title("0.1 User Defined")
axs[0,0].imshow(median_filter(a1,3),"gray")
axs[0,1].set_title("0.2 User Defined")
axs[0,1].imshow(median_filter(a2,3),"gray")
axs[0,2].set_title("0.3 User Defined")
axs[0,2].imshow(median_filter(a3,3),"gray")
axs[0,3].set_title("0.4 User Defined")
axs[0,3].imshow(median_filter(a4,3),"gray")
axs[0,4].set_title("0.5 User Defined")
axs[0,4].imshow(median_filter(a5,3),"gray")
axs[1,0].set_title("0.6 User Defined")
axs[1,0].imshow(median_filter(a6,3),"gray")
axs[1,1].set_title("0.7 User Defined")
axs[1,1].imshow(median_filter(a7,3),"gray")
axs[1,2].set_title("0.8 User Defined")
axs[1,2].imshow(median_filter(a8,3),"gray")
axs[1,3].set_title("0.9 User Defined")
axs[1,3].imshow(median_filter(a9,3),"gray")
axs[1,4].set_title("1.0 User Defined")
axs[1,4].imshow(median_filter(a10,3),"gray")

#Inbuilt
axs[2,0].set_title("0.1 In-Built")
axs[2,0].imshow(cv2.medianBlur(a1,3),"gray")
axs[2,1].set_title("0.2 In-Built")
axs[2,1].imshow(cv2.medianBlur(a2,3),"gray")
axs[2,2].set_title("0.3 In-Built")
axs[2,2].imshow(cv2.medianBlur(a3,3),"gray")
axs[2,3].set_title("0.4 In-Built")
axs[2,3].imshow(cv2.medianBlur(a4,3),"gray")
axs[2,4].set_title("0.5 In-Built")
axs[2,4].imshow(cv2.medianBlur(a5,3),"gray")
axs[3,0].set_title("0.6 In-Built")
axs[3,0].imshow(cv2.medianBlur(a6,3),"gray")
axs[3,1].set_title("0.7 In-Built")
axs[3,1].imshow(cv2.medianBlur(a7,3),"gray")
axs[3,2].set_title("0.8 In-Built")
axs[3,2].imshow(cv2.medianBlur(a8,3),"gray")
axs[3,3].set_title("0.9 In-Built")
axs[3,3].imshow(cv2.medianBlur(a9,3),"gray")
axs[3,4].set_title("1.0 In-Built")
```

```
axs[3,4].imshow(cv2.medianBlur(a10,3),"gray")

plt.setp(plt.gcf().get_axes(), xticks=[], yticks=[])
```

Out[13]: []

5x5

In [14]:

```
#5x5 Median Filtering

fig,axs = plt.subplots(4,5,figsize=(20,20))

plt.suptitle("5x5 Median Filtering",fontsize=30)

axs[0,0].set_title("0.1 User Defined")
axs[0,0].imshow(median_filter(a1,5),"gray")
axs[0,1].set_title("0.2 User Defined")
axs[0,1].imshow(median_filter(a2,5),"gray")
axs[0,2].set_title("0.3 User Defined")
axs[0,2].imshow(median_filter(a3,5),"gray")
axs[0,3].set_title("0.4 User Defined")
axs[0,3].imshow(median_filter(a4,5),"gray")
axs[0,4].set_title("0.5 User Defined")
axs[0,4].imshow(median_filter(a5,5),"gray")
axs[1,0].set_title("0.6 User Defined")
axs[1,0].imshow(median_filter(a6,5),"gray")
axs[1,1].set_title("0.7 User Defined")
axs[1,1].imshow(median_filter(a7,5),"gray")
axs[1,2].set_title("0.8 User Defined")
axs[1,2].imshow(median_filter(a8,5),"gray")
axs[1,3].set_title("0.9 User Defined")
axs[1,3].imshow(median_filter(a9,5),"gray")
axs[1,4].set_title("1.0 User Defined")
axs[1,4].imshow(median_filter(a10,5),"gray")

#Inbuilt
axs[2,0].set_title("0.1 In-Built")
axs[2,0].imshow(cv2.medianBlur(a1,5),"gray")
axs[2,1].set_title("0.2 In-Built")
axs[2,1].imshow(cv2.medianBlur(a2,5),"gray")
axs[2,2].set_title("0.3 In-Built")
axs[2,2].imshow(cv2.medianBlur(a3,5),"gray")
axs[2,3].set_title("0.4 In-Built")
axs[2,3].imshow(cv2.medianBlur(a4,5),"gray")
axs[2,4].set_title("0.5 In-Built")
axs[2,4].imshow(cv2.medianBlur(a5,5),"gray")
axs[3,0].set_title("0.6 In-Built")
axs[3,0].imshow(cv2.medianBlur(a6,5),"gray")
axs[3,1].set_title("0.7 In-Built")
axs[3,1].imshow(cv2.medianBlur(a7,5),"gray")
axs[3,2].set_title("0.8 In-Built")
axs[3,2].imshow(cv2.medianBlur(a8,5),"gray")
axs[3,3].set_title("0.9 In-Built")
axs[3,3].imshow(cv2.medianBlur(a9,5),"gray")
axs[3,4].set_title("1.0 In-Built")
axs[3,4].imshow(cv2.medianBlur(a10,5),"gray")

plt.setp(plt.gcf().get_axes(), xticks=[], yticks=[])
```

Out[14]: []

5x5 Median Filtering

7x7

In [16]:

```
#7x7 Median Filtering

fig,axs = plt.subplots(4,5,figsize=(20,20))

plt.suptitle("7x7 Median Filtering",fontsize=30)

axs[0,0].set_title("0.1 User Defined")
axs[0,0].imshow(median_filter(a1,7),"gray")
axs[0,1].set_title("0.2 User Defined")
axs[0,1].imshow(median_filter(a2,7),"gray")
axs[0,2].set_title("0.3 User Defined")
axs[0,2].imshow(median_filter(a3,7),"gray")
axs[0,3].set_title("0.4 User Defined")
axs[0,3].imshow(median_filter(a4,7),"gray")
axs[0,4].set_title("0.5 User Defined")
axs[0,4].imshow(median_filter(a5,7),"gray")
axs[1,0].set_title("0.6 User Defined")
axs[1,0].imshow(median_filter(a6,7),"gray")
axs[1,1].set_title("0.7 User Defined")
axs[1,1].imshow(median_filter(a7,7),"gray")
axs[1,2].set_title("0.8 User Defined")
axs[1,2].imshow(median_filter(a8,7),"gray")
axs[1,3].set_title("0.9 User Defined")
axs[1,3].imshow(median_filter(a9,7),"gray")
axs[1,4].set_title("1.0 User Defined")
axs[1,4].imshow(median_filter(a10,7),"gray")

#Inbuilt
axs[2,0].set_title("0.1 In-Built")
axs[2,0].imshow(cv2.medianBlur(a1,7),"gray")
axs[2,1].set_title("0.2 In-Built")
axs[2,1].imshow(cv2.medianBlur(a2,7),"gray")
axs[2,2].set_title("0.3 In-Built")
axs[2,2].imshow(cv2.medianBlur(a3,7),"gray")
axs[2,3].set_title("0.4 In-Built")
axs[2,3].imshow(cv2.medianBlur(a4,7),"gray")
axs[2,4].set_title("0.5 In-Built")
axs[2,4].imshow(cv2.medianBlur(a5,7),"gray")
axs[3,0].set_title("0.6 In-Built")
axs[3,0].imshow(cv2.medianBlur(a6,7),"gray")
axs[3,1].set_title("0.7 In-Built")
axs[3,1].imshow(cv2.medianBlur(a7,7),"gray")
axs[3,2].set_title("0.8 In-Built")
axs[3,2].imshow(cv2.medianBlur(a8,7),"gray")
axs[3,3].set_title("0.9 In-Built")
axs[3,3].imshow(cv2.medianBlur(a9,7),"gray")
axs[3,4].set_title("1.0 In-Built")
axs[3,4].imshow(cv2.medianBlur(a10,7),"gray")

plt.setp(plt.gcf().get_axes(), xticks=[], yticks=[])
```

Out[16]: []

7x7 Median Filtering

Question 2

a. Consider the image the attached named as hdraw.png and crop each of the characters from the image and consider that as the sub-image. Find the location of the sub-image in the original by using correlation.

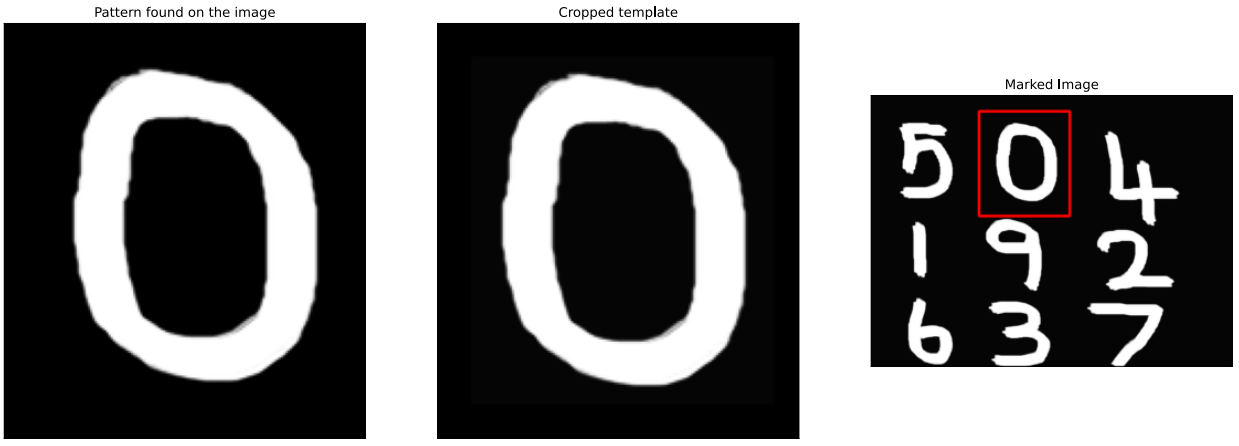
In [11]:
```
imgRef = mpimg.imread('hdraw.png')
imgRef = rgb2gray(imgRef)
```

In [1]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import cv2
import math
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity

def rgb2gray(rgb):
    return np.dot(rgb[...,:3], [0.299, 0.587, 0.144])

def crossCorrelation(p,q):
    p,q = p[0], q[0]
    sum = 0
    for i in range(len(p)):
        sum += p[i] * q[i]
    return sum


imgRef = mpimg.imread('hdraw.png')
imgRef = rgb2gray(imgRef)
# plt.imshow(imgRef,cmap="gray")
# plt.show()

img = mpimg.imread('0.png')
img = rgb2gray(img)
# plt.imshow(img,cmap="gray")
# plt.show()
dim = img.shape

print(imgRef.shape)
print("before padding :", len(img), len(img[0]))
img = np.array(img)

p, q = int(1.2*len(img)), int(1.2*len(img[0]))
print("after padding :", p,q)
padded = np.zeros((p,q))
ind = int(0.1*len(img))
padded[ind:ind + img.shape[0], ind:ind + img.shape[1]] = img
img = padded

imgVector = [img[i,j] for j in range(len(img[0])) for i in range(len(img))]
# print(imgVector)

maxi, maxi1 = -1, -1
ans = []
for x in range(0, len(imgRef) - len(img), 10):
    for y in range(0, len(imgRef[0]) - len(img[0]), 5):
        # print("hi")
        temp = imgRef[x:x+len(img), y:y+len(img[0])]
        tempVector = [temp[i,j] for j in range(len(temp[0])) for i in range(len(temp))]
        a = np.array(imgVector).reshape(1,len(imgVector))
        a = a - np.mean(a)
        b = np.array(tempVector).reshape(1,len(tempVector))
        b = b - np.mean(b)
        out = crossCorrelation(a,b) #user-defined
        mod1 = np.linalg.norm(a)
        mod2 = np.linalg.norm(b)
        out1 = cosine_similarity(X=a, Y=b, dense_output=True) * (mod1 * mod2) #built-in function
        maxi1 = out1 if out1 > maxi1 else maxi1
        if out > maxi:
            maxi = out
            ans =[]
            ans.append(temp)
            ans.append(img)
            X,Y = x,y

print("max_correlation (user-defined) : ", maxi)
print("max_correlation (built-in) : ", maxi1)
```

```
fig, axs = plt.subplots(1, 3,figsize=(20,20))


imgRef = mpimg.imread('hdraw.png')

axs[0].set_title("Pattern found on the image")
axs[0].imshow(ans[0],cmap="gray")

axs[1].set_title("Cropped template")
axs[1].imshow(ans[1],cmap="gray")

imgRef = cv2.rectangle(imgRef, (Y, X), (Y + q, X + p), (0, 0,255), 4)
imgRef = cv2.cvtColor(imgRef, cv2.COLOR_BGR2RGB)

axs[2].set_title("Marked Image")
axs[2].imshow(imgRef)

#remove Ticks
axs[0].set_xticks([])
axs[0].set_yticks([])
axs[1].set_xticks([])
axs[1].set_yticks([])
axs[2].set_xticks([])
axs[2].set_yticks([])
```

```
(480, 640)
before padding : 154 134
after padding : 184 160
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
max_correlation (user-defined) :  4640.994977985809
max_correlation (built-in) :  [[4640.99497799]]
```

Out[1]: []



b. Download Lena color image convert it to grayscale image and crop the left eye of Lena as sub-image and do the cross-correlation (Normalized correlation) to find the location of the left eye in the original image.

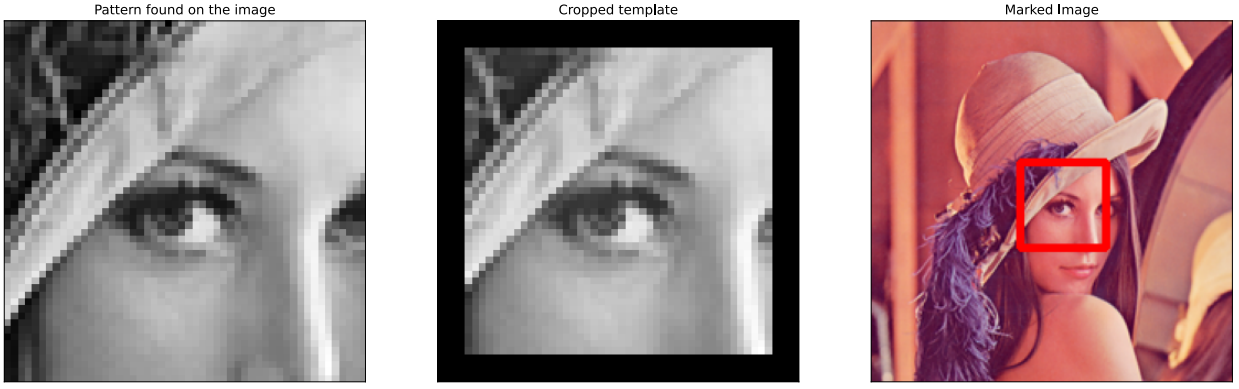
In [35]:

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import cv2
import math
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity

def rgb2gray(rgb):
    return np.dot(rgb[...,:3], [0.299, 0.587, 0.144])

def normalizedCrossCorrelation(p,q):
    p,q = p[0], q[0]
    sum, z1, z2 = 0, 0, 0
    for i in range(len(p)):
        sum += p[i] * q[i]
        z1 += p[i] * p[i]
        z2 += q[i] * q[i]
    return sum / math.sqrt(z1*z2)


imgRef = mpimg.imread('lena.png')
imgRef = rgb2gray(imgRef)

img = mpimg.imread('lenaLeft.png')
img = rgb2gray(img)

dim = img.shape

print(imgRef.shape)
print("before padding :", len(img), len(img[0]))
img = np.array(img)

p, q = int(1.2*len(img)), int(1.2*len(img[0]))
print("after padding :", p,q)
padded = np.zeros((p,q))
ind = int(0.1*len(img))
padded[ind:ind + img.shape[0], ind:ind + img.shape[1]] = img
img = padded

imgVector = [img[i,j] for j in range(len(img[0])) for i in range(len(img))]
# print(imgVector)

maxi, maxi1 = -1, -1
ans = []
for x in range(0, len(imgRef) - len(img), 2):
    for y in range(0, len(imgRef[0]) - len(img[0]), 2):
        # print("Reached check 2, bug after this")
        temp = imgRef[x:x+len(img), y:y+len(img[0])]
        tempVector = [temp[i,j] for j in range(len(temp[0])) for i in range(len(temp))]
        a = np.array(imgVector).reshape(1,len(imgVector))
        a = a - np.mean(a)
        b = np.array(tempVector).reshape(1,len(tempVector))
        b = b - np.mean(b)
        temp1,temp2 = a, b
        mod1 = np.linalg.norm(a)
        mod2 = np.linalg.norm(b)
        a /= mod1
        b /= mod2
        out = normalizedCrossCorrelation(a,b) #user-defined
        out1 = cosine_similarity(X=temp1, Y=temp2, dense_output=True) #built-in function
        maxi1 = out1 if out1 > maxi1 else maxi1
        if out > maxi:
            maxi = out
            ans =[]
            ans.append(temp)
            ans.append(img)
            X,Y = x,y


print("max_correlation (user-defined) :", maxi)
print("max_correlation (built-in) :", maxi1)


fig, axs = plt.subplots(1, 3,figsize=(20,20))


imgRef = mpimg.imread('lena.png')

axs[0].set_title("Pattern found on the image")
axs[0].imshow(ans[0],cmap="gray")
```

```
axs[1].set_title("Cropped template")
axs[1].imshow(ans[1],cmap="gray")

imgRef = cv2.rectangle(imgRef, (Y, X), (Y + q, X + p), (255, 0, 0), 4)

axs[2].set_title("Marked Image")
axs[2].imshow(imgRef)

#remove Ticks
axs[0].set_xticks([])
axs[0].set_yticks([])
axs[1].set_xticks([])
axs[1].set_yticks([])
axs[2].set_xticks([])
axs[2].set_yticks([])
```

```
(220, 220)
before padding : 44 44
after padding : 52 52
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
max_correlation (user-defined) : 0.5194785902019393
max_correlation (built-in) : [[0.51947859]]
```

Out[35]: []



Question 3

Write a function to implement FFT for 1D signal.

In [1]:

```
import numpy as np

x = np.random.randint(8,16,8)

def FFT(x):
    F=[]
    x = np.asarray(x, dtype=float)
    N = x.shape[0]
    for U in range(0,N):
        sum=0
        for y in range(0,N):
            sum=sum+(x[y]*(np.exp(-2j * np.pi * U * y / N)))
        F.append(sum)
    return(F)

x = np.random.random(10)
x = np.asarray(x)
print("\nUser-Defined\n",FFT(x))
print("\nInbuilt\n",np.fft.fft(x) )
print("\nCompare\n")
print(np.allclose(np.fft.fft(x),FFT(x)))
```

```
User-Defined
 [(5.761206968598533+0j), (1.032707566865341-0.27842968690584796j), (0.2425796146191278+1.0696397113319729j),
(0.059402470698636245+0.41088191511026056j), (0.5478046726616833-0.767787455380745j), (0.03898315876891989-4.257
9360917508727e-16j), (0.5478046726616828+0.76778745538074 42j), (0.05940247069863419-0.41088191511026073j), (0.24
25796146191304-1.0696397113319733j), (1.032707566865344+0.2784296869058511j)]

Inbuilt
 [5.76120697+0.00000000e+00j 1.03270757-2.78429687e-01j
 0.24257961+1.06963971e+00j 0.05940247+4.10881915e-01j
 0.54780467-7.67787455e-01j 0.03898316-2.77555756e-17j
 0.54780467+7.67787455e-01j 0.05940247-4.10881915e-01j
 0.24257961-1.06963971e+00j 1.03270757+2.78429687e-01j]

Compare

True
```

Question 4

Implement DFT function for an image using the FFT for 1D signal using question 3.

In [65]:

```
import numpy as np
s=8

x = np.random.random(s)


def FFT(x):
    F=[]
    #x = np.asarray(x, dtype=float)
    N = x.shape[0]
    for U in range(0,N):
        sum=0
        for y in range(0,N):
            sum=sum+(x[y]*(np.exp(-2j * np.pi * U * y / N)))
        F.append(sum)
    return(F)

x = np.random.rand(s,s)
F2=[]
Ftemp=np.zeros((s,s),dtype=complex)
temp=[]

M = x.shape[0]


for i in range(0,M):
    temp.append(FFT(x[i]))
temp = np.array(temp)


N= x.shape[1]
for i in range(0,N):
    #print(FFT(temp[:,i]))
    Ftemp[:,i] = FFT(temp[:,i])

print("Built In\n")
print(np.fft.fft2(x))
print("\nUser Defined\n")
print(Ftemp)
print("\n\n")
print("Compare\n")
print(np.allclose(np.fft.fft2(x),Ftemp))
```

```
Built In

[[ 3.16895924e+01+0.j          4.62229621e-01+1.26934763j
  -4.77468750e+00-1.36966719j  1.50280878e-01+1.43253573j
  -2.56076164e+00+0.j          1.50280878e-01-1.43253573j
  -4.77468750e+00+1.36966719j  4.62229621e-01-1.26934763j]
 [ 1.13225683e+00-0.66207923j -2.61001173e+00+0.77791159j
   1.25834726e+00+0.48130899j  1.85202556e+00-1.22233445j
  -4.01355587e-01+1.77290412j  3.88922669e-01-0.84795156j
  -1.85719953e-01-0.69425614j  3.97391259e-01-0.61413599j]
 [ 1.29566814e+00+0.06537081j -4.24287213e-01-1.61003218j
   6.45194295e-01+0.94668932j  1.56696142e-01-0.54343372j
  -1.56548262e+00-0.41206106j -5.72979324e-01-0.14403723j
   1.23772774e+00+0.63361469j  3.96923863e+00-2.3864684j ]
 [ 8.87352904e-01-1.05967969j  1.03817268e+00+0.4579616j
   6.70384487e-01+0.11030579j  3.24149781e+00+1.61542013j
   3.89281939e+00+0.32766713j -1.40074952e+00-0.54304404j
   1.36442671e-01-0.439433j   -2.48177839e-02+0.64328561j]
 [ 2.10515840e+00+0.j         -3.48101492e-01+5.19682363j
  -1.13885598e+00+3.49980295j  2.68673862e+00-0.56486925j
  -5.53309870e-01+0.j           2.68673862e+00+0.56486925j
  -1.13885598e+00-3.49980295j -3.48101492e-01-5.19682363j]
 [ 8.87352904e-01+1.05967969j -2.48177839e-02-0.64328561j
   1.36442671e-01+0.439433j   -1.40074952e+00+0.54304404j
   3.89281939e+00-0.32766713j  3.24149781e+00-1.61542013j
   6.70384487e-01-0.11030579j  1.03817268e+00-0.4579616j ]
 [ 1.29566814e+00-0.06537081j  3.96923863e+00+2.3864684j
   1.23772774e+00-0.63361469j -5.72979324e-01+0.14403723j
  -1.56548262e+00+0.41206106j  1.56696142e-01+0.54343372j
   6.45194295e-01-0.94668932j -4.24287213e-01+1.61003218j]
 [ 1.13225683e+00+0.66207923j  3.97391259e-01+0.61413599j
  -1.85719953e-01+0.69425614j  3.88922669e-01+0.84795156j
  -4.01355587e-01-1.77290412j  1.85202556e+00+1.22233445j
   1.25834726e+00-0.48130899j -2.61001173e+00-0.77791159j]]
```

User Defined

```
[[ 3.16895924e+01+0.00000000e+00j  4.62229621e-01+1.26934763e+00j
  -4.77468750e+00-1.36966719e+00j  1.50280878e-01+1.43253573e+00j
  -2.56076164e+00-2.84850959e-15j  1.50280878e-01-1.43253573e+00j
  -4.77468750e+00+1.36966719e+00j  4.62229621e-01-1.26934763e+00j]
 [ 1.13225683e+00-6.62079230e-01j -2.61001173e+00+7.77911590e-01j
   1.25834726e+00+4.81308985e-01j  1.85202556e+00-1.22233445e+00j
  -4.01355587e-01+1.77290412e+00j  3.88922669e-01-8.47951561e-01j
  -1.85719953e-01-6.94256135e-01j  3.97391259e-01-6.14135985e-01j]
 [ 1.29566814e+00+6.53708067e-02j -4.24287213e-01-1.61003218e+00j
   6.45194295e-01+9.46689321e-01j  1.56696142e-01-5.43433719e-01j
  -1.56548262e+00-4.12061056e-01j -5.72979324e-01-1.44037226e-01j
   1.23772774e+00+6.33614689e-01j  3.96923863e+00-2.38646840e+00j]
 [ 8.87352904e-01-1.05967969e+00j  1.03817268e+00+4.57961599e-01j
   6.70384487e-01+1.10305791e-01j  3.24149781e+00+1.61542013e+00j
   3.89281939e+00+3.27667135e-01j -1.40074952e+00-5.43044038e-01j
   1.36442671e-01-4.39432998e-01j -2.48177839e-02+6.43285612e-01j]
 [ 2.10515840e+00-1.10522600e-15j -3.48101492e-01+5.19682363e+00j
  -1.13885598e+00+3.49980295e+00j  2.68673862e+00-5.64869251e-01j
  -5.53309870e-01-1.49078348e-15j  2.68673862e+00+5.64869251e-01j
  -1.13885598e+00-3.49980295e+00j -3.48101492e-01-5.19682363e+00j]
 [ 8.87352904e-01+1.05967969e+00j -2.48177839e-02-6.43285612e-01j
   1.36442671e-01+4.39432998e-01j -1.40074952e+00+5.43044038e-01j
   3.89281939e+00-3.27667135e-01j  3.24149781e+00-1.61542013e+00j
   6.70384487e-01-1.10305791e-01j  1.03817268e+00-4.57961599e-01j]
 [ 1.29566814e+00-6.53708067e-02j  3.96923863e+00+2.38646840e+00j
   1.23772774e+00-6.33614689e-01j -5.72979324e-01+1.44037226e-01j
  -1.56548262e+00+4.12061056e-01j  1.56696142e-01+5.43433719e-01j
   6.45194295e-01-9.46689321e-01j -4.24287213e-01+1.61003218e+00j]
 [ 1.13225683e+00+6.62079230e-01j  3.97391259e-01+6.14135985e-01j
  -1.85719953e-01+6.94256135e-01j  3.88922669e-01+8.47951561e-01j
  -4.01355587e-01-1.77290412e+00j  1.85202556e+00+1.22233445e+00j
   1.25834726e+00-4.81308985e-01j -2.61001173e+00-7.77911590e-01j]]
```

Compare

True

Question 5

Consider the images of lena and dog images attached. Find phase and magnitude of the dog and lena images using DFT function implemented in question 4.

In [20]:

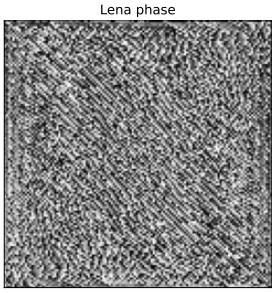
```
import numpy as np
import cv2
import cmath
import matplotlib.pyplot as plt

lena = cv2.imread("lena.jpeg",0)
lena = np.array(lena)
dog = cv2.imread("dog.png",0)
dog = np.array(dog)

#Function for Fourier Transform
def FFT(x):
    F=[]
    #x = np.asarray(x, dtype=float)
    N = x.shape[0]
    for U in range(0,N):
        sum=0
        for y in range(0,N):
            sum=sum+(x[y]*(np.exp(-2j * np.pi * U * y / N)))
        F.append(sum)
    return(F)

row_FFT=[] #To store result of row-wise 1-dog FFT
col_FFT = np.zeros((lena.shape[0],lena.shape[1]),dtype=complex) #To store result of col-wise 1-dog FFT

M = lena.shape[0]
#row-wise FFT
for i in range(0,M):
    row_FFT.append(FFT(lena[i]))
row_FFT = np.array(row_FFT)

#col-wise FFT
N = lena.shape[1]
for i in range(0,N):
    col_FFT[:,i] = FFT(row_FFT[:,i])



#separataing phase and magnitude of Lena
lena_phase = []
lena_mag = []

lena_phase=np.angle(col_FFT)
lena_mag=np.log(np.abs(col_FFT))

lena_phase_inbuilt=np.angle(np.fft.fft2(lena))
lena_mag_inbuilt=np.log(np.abs(np.fft.fft2(lena)))

row_FFT=[]#To store result of row-wise 1-dog FFT
col_FFT = np.zeros((dog.shape[0],dog.shape[1]),dtype=complex) #To store result of col-wise 1-dog FFT

#row-wise  FFT dog
M = dog.shape[0]
# print(dog.shape)
for i in range(0,M):
    row_FFT.append(FFT(dog[i]))

row_FFT = np.array(row_FFT)

#col-wise FFT dog
N = dog.shape[1]
for i in range(0,N):
    col_FFT[:,i] = FFT(row_FFT[:,i])

#separataing phase and magnitude of dog
dog_phase = []
dog_mag = []

dog_phase=np.angle(col_FFT)
dog_mag=np.log(np.abs(col_FFT))

dog_phase_inbuilt=np.angle(np.fft.fft2(dog))
dog_mag_inbuilt=np.log(np.abs(np.fft.fft2(dog)))
```

```
fig, axs = plt.subplots(2, 4,figsize=(20,20))

#plot Lena
axs[0,0].set_title('Lena phase')
axs[0,0].imshow(lena_phase,cmap='gray')
axs[0,1].set_title('Lena phase inbuilt')
axs[0,1].imshow(lena_phase_inbuilt,cmap='gray')
axs[0,2].set_title('Lena Mag')
axs[0,2].imshow(lena_mag,cmap='gray')
axs[0,3].set_title('Lena Mag inbuilt')
axs[0,3].imshow(lena_mag_inbuilt,cmap='gray')

#plot dog
axs[1,0].set_title('Dog phase')
axs[1,0].imshow(dog_phase,cmap='gray')
axs[1,1].set_title('Dog phase inbuilt')
axs[1,1].imshow(dog_phase_inbuilt,cmap='gray')
axs[1,2].set_title('Dog Mag')
axs[1,2].imshow(dog_mag,cmap='gray')
axs[1,3].set_title('Dog Mag inbuilt')
axs[1,3].imshow(dog_mag_inbuilt,cmap='gray')

#remove Lines
axs[0,0].set_xticks([])
axs[0,0].set_yticks([])
axs[0,1].set_xticks([])
axs[0,1].set_yticks([])
axs[0,2].set_xticks([])
axs[0,2].set_yticks([])
axs[0,3].set_xticks([])
axs[0,3].set_yticks([])
axs[1,0].set_xticks([])
axs[1,0].set_yticks([])
axs[1,1].set_xticks([])
axs[1,1].set_yticks([])
axs[1,2].set_xticks([])
axs[1,2].set_yticks([])
axs[1,3].set_xticks([])
axs[1,3].set_yticks([])

plt.suptitle('Lena and Dog Phase',fontsize=30)
```

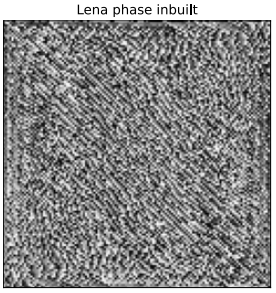
Out[20]: Text(0.5, 0.98, 'Lena and Dog Phase')

Lena and Dog Phase

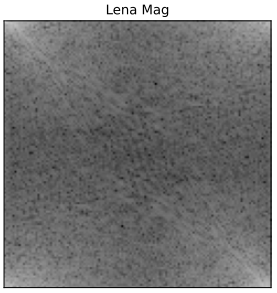
Lena phase



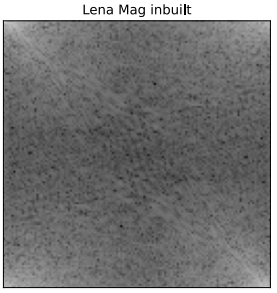
Lena phase inbuilt



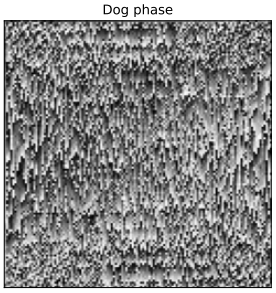
Lena Mag



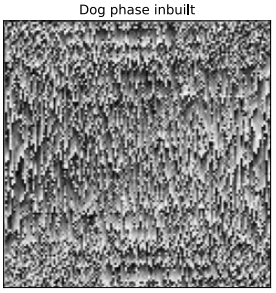
Lena Mag inbuilt



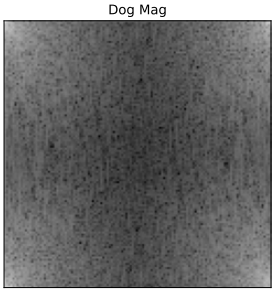
Dog phase



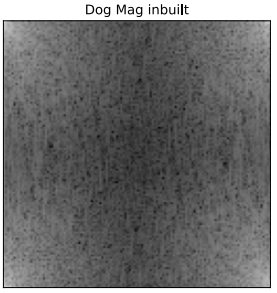
Dog phase inbuilt



Dog Mag



Dog Mag inbuilt



Question 6

Swap phase of the dog image and magnitude of the lena image and display the output.

In [18]:

```
import numpy as np
import cv2
import cmath
import matplotlib.pyplot as plt

lena = cv2.imread("lena.jpeg",0)
lena = np.array(lena)
dog = cv2.imread("dog.png",0)
dog = np.array(dog)

def FFT(x):
    F=[]
    #x = np.asarray(x, dtype=float)
    N = x.shape[0]
    for U in range(0,N):
        sum=0
        for y in range(0,N):
            sum=sum+(x[y]*(np.exp(-2j * np.pi * U * y / N)))
        F.append(sum)
    return(F)

def IFFT(x):
    F=[]
    #x = np.asarray(x, dtype=float)
    N = x.shape[0]
    for U in range(0,N):
        sum=0
        for y in range(0,N):
            sum=sum+(x[y]*(np.exp(2j * np.pi * U * y / N)))
        F.append(sum/N)
    return(F)
row_FFT=[] #To store result of row-wise 1-dog FFT
col_FFT = np.zeros((lena.shape[0],lena.shape[1]),dtype=complex) #To store result of col-wise 1-dog FFT

M = lena.shape[0]
#row-wise FFT
for i in range(0,M):
    row_FFT.append(FFT(lena[i]))
row_FFT = np.array(row_FFT)

#col-wise FFT
N = lena.shape[1]
for i in range(0,N):
    col_FFT[:,i] = FFT(row_FFT[:,i])



#separataing phase and magnitude of Lena
lena_phase = []
lena_mag = []

lena_phase=np.angle(col_FFT)
lena_mag=np.log(np.abs(col_FFT))

lena_phase_inbuilt=np.angle(np.fft.fft2(lena))
lena_mag_inbuilt=np.log(np.abs(np.fft.fft2(lena)))

row_FFT=[]#To store result of row-wise 1-dog FFT
col_FFT = np.zeros((dog.shape[0],dog.shape[1]),dtype=complex) #To store result of col-wise 1-dog FFT

#row-wise  FFT dog
M = dog.shape[0]
print(dog.shape)
for i in range(0,M):
    row_FFT.append(FFT(dog[i]))

row_FFT = np.array(row_FFT)

#col-wise FFT dog
N = dog.shape[1]
for i in range(0,N):
    col_FFT[:,i] = FFT(row_FFT[:,i])

#separataing phase and magnitude of dog
dog_phase = []
```

```
dog_mag = []

dog_phase=np.angle(col_FFT)
dog_mag=np.log(np.abs(col_FFT))

dog_phase_inbuilt=np.angle(np.fft.fft2(dog))
dog_mag_inbuilt=np.log(np.abs(np.fft.fft2(dog)))


#combining phase and magnitude of lena and dog resp.
combined=np.multiply(dog_mag,np.exp(1j*lena_phase))


#IFFT to get output
M = combined.shape[0]
row_IFFT=[]
col_IFFT = np.zeros((combined.shape[0],combined.shape[1]),dtype=complex)
for i in range(0,M):
    row_IFFT.append(IFFT(combined[i]))

col_IFFT = np.array(row_IFFT)
FFtemp = np.zeros((combined.shape[0],combined.shape[1]),dtype=complex)

N = combined.shape[1]
for i in range(0,N):
    FFtemp[:,i] = IFFT(col_IFFT[:,i])
IFFTcombined=np.asarray(FFtemp)

fig, axs = plt.subplots(1, 2,figsize=(20,20))

axs[0].set_title('Lena phase + Dog Mag')
axs[0].imshow( np.real(IFFTcombined),cmap='gray')

axs[1].set_title('Lena phase + Dog Mag inbuilt')
axs[1].imshow(np.real(np.fft.ifft2(combined)),cmap='gray')

axs[0].set_xticks([])
axs[0].set_yticks([])
axs[1].set_xticks([])
axs[1].set_yticks([])

plt.suptitle('Lena phase + Dog Magnitude',fontsize=30)
```

```
(128, 128)
```

Out[18]:
```
Text(0.5, 0.98, 'Lena phase + Dog Magnitude')
```

Lena phase + Dog Magnitude

Lena phase + Dog Mag

Lena phase + Dog Mag inbuilt

Question 7

Swap phase of the lena image and magnitude of the dog image ad display the output

In [19]:

```
import numpy as np
import cv2
import cmath
import matplotlib.pyplot as plt

lena = cv2.imread("lena.jpeg",0)
lena = np.array(lena)
dog = cv2.imread("dog.png",0)
dog = np.array(dog)

def FFT(x):
    F=[]
    #x = np.asarray(x, dtype=float)
    N = x.shape[0]
    for U in range(0,N):
        sum=0
        for y in range(0,N):
            sum=sum+(x[y]*(np.exp(-2j * np.pi * U * y / N)))
        F.append(sum)
    return(F)

def IFFT(x):
    F=[]
    #x = np.asarray(x, dtype=float)
    N = x.shape[0]
    for U in range(0,N):
        sum=0
        for y in range(0,N):
            sum=sum+(x[y]*(np.exp(2j * np.pi * U * y / N)))
        F.append(sum/N)
    return(F)
row_FFT=[] #To store result of row-wise 1-dog FFT
col_FFT = np.zeros((lena.shape[0],lena.shape[1]),dtype=complex) #To store result of col-wise 1-dog FFT

M = lena.shape[0]
#row-wise FFT
for i in range(0,M):
    row_FFT.append(FFT(lena[i]))
row_FFT = np.array(row_FFT)

#col-wise FFT
N = lena.shape[1]
for i in range(0,N):
    col_FFT[:,i] = FFT(row_FFT[:,i])



#separataing phase and magnitude of Lena
lena_phase = []
lena_mag = []

lena_phase=np.angle(col_FFT)
lena_mag=np.log(np.abs(col_FFT))

lena_phase_inbuilt=np.angle(np.fft.fft2(lena))
lena_mag_inbuilt=np.log(np.abs(np.fft.fft2(lena)))

row_FFT=[]#To store result of row-wise 1-dog FFT
col_FFT = np.zeros((dog.shape[0],dog.shape[1]),dtype=complex) #To store result of col-wise 1-dog FFT

#row-wise  FFT dog
M = dog.shape[0]
print(dog.shape)
for i in range(0,M):
    row_FFT.append(FFT(dog[i]))

row_FFT = np.array(row_FFT)

#col-wise FFT dog
N = dog.shape[1]
for i in range(0,N):
    col_FFT[:,i] = FFT(row_FFT[:,i])

#separataing phase and magnitude of dog
dog_phase = []
```

```
dog_mag = []

dog_phase=np.angle(col_FFT)
dog_mag=np.log(np.abs(col_FFT))

dog_phase_inbuilt=np.angle(np.fft.fft2(dog))
dog_mag_inbuilt=np.log(np.abs(np.fft.fft2(dog)))


#combining phase and magnitude of dog and lena resp.
combined=np.multiply(lena_mag,np.exp(1j*dog_phase))


#IFFT to get output
M = combined.shape[0]
row_IFFT=[]
col_IFFT = np.zeros((combined.shape[0],combined.shape[1]),dtype=complex)

#performing row-wise 1-d IFFT
for i in range(0,M):
    row_IFFT.append(IFFT(combined[i]))
row_IFFT = np.array(row_IFFT)
print(row_IFFT.shape)
N = combined.shape[1]
for i in range(0,N):
    col_IFFT[:,i] = IFFT(row_IFFT[:,i])
IFFTcombined=np.asarray(col_IFFT)

fig, axs = plt.subplots(1, 2,figsize=(20,20))

axs[0].set_title('Lena Mag + Dog Phase')
axs[0].imshow( np.real(IFFTcombined),cmap='gray')

axs[1].set_title('Lena mag + Dog phase inbuilt')
axs[1].imshow(np.real(np.fft.ifft2(combined)),cmap='gray')

axs[0].set_xticks([])
axs[0].set_yticks([])
axs[1].set_xticks([])
axs[1].set_yticks([])

plt.suptitle('Lena Magnitude + Dog Phase',fontsize=30)
```

```
(128, 128)
(128, 128)
```

Out[19]: Text(0.5, 0.98, 'Lena Magnitude + Dog Phase')

Lena Magnitude + Dog Phase

Lena Mag + Dog Phase

Lena mag + Dog phase inbuilt