Oppgave 2

In [1]:

```
from skimage import io, util, color
from scipy import fftpack, ndimage
import numpy

import ipywidgets as widgets
from ipywidgets import interactive
```

Her velger man bildet som skal prosesseres, jeg har lagt inn bildene fra oppgaven.

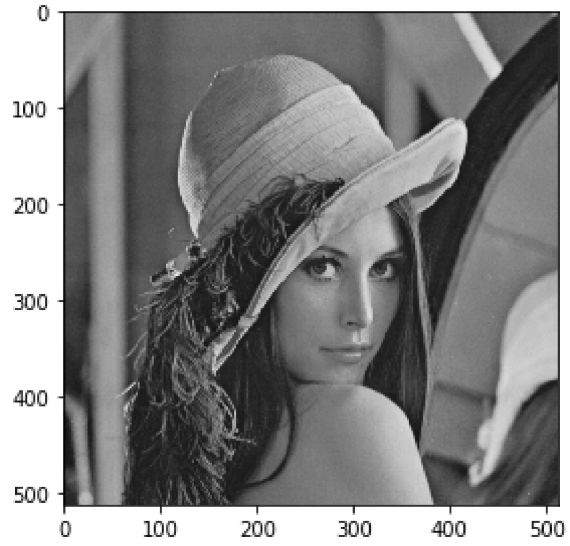
In [2]:

```
def readAndShowImage(image):
    im = io.imread(image, as_gray=True)
    io.imshow(im)
    io.show()

inputImage = widgets.Dropdown(
    options={
        'Lena':'lena.png',
        'Barbara':'barbara.png'
    },
    description='Bilde: ',
    disabled=False,
)

interactive(readAndShowImage, image=inputImage)
```

Bilde: Lena



Funksjoner som trengs for frekvensfiltrering.

dist funksjonen brukes for å genere butterworth filtrene. fourrierFilter tar imot ett bilde og ett filter som skal bruker for frekvensfiltrering.

In [3]:

```
def dist(i,j,height,width):
    center_i, center_j = (height/2,width/2)
    return ((i-center_i)**2 + (j-center_j)**2)**0.5

def butterworth(height, width, cutoff, order):
    filtr = numpy.zeros((height,width))

    for i in range(height):
        for j in range(width):
            filtr[i,j] = 1 / (1 + (cutoff/ max(dist(i,j,height,width),1) )**(2*order))

    return filtr

def butterworthLowPass(height, width, cutoff, order):
    filtr = numpy.zeros((height,width))

    for i in range(height):
        for j in range(width):
            filtr[i,j] = 1 / (1 + ( max(dist(i,j,height,width),1) / cutoff )**(2*order))

    return filtr

def fourrierFilter(im, filtr):
    # Retrieve the image dimensions
    height,width = im.shape

    #finding P and Q
    padded_height = height*2
    padded_width = width*2

    #Padding image width 0s
    padded = numpy.zeros((padded_height,padded_width))
    padded[0:height,0:width] = im

    # Create an image to store the transformed image
    image_trans = numpy.zeros(padded.shape)
    # Transform each pixel in the image by multiplying with -1**(i+j) (see
    # slides/book) to translate the fourier transformed image to height/2 width/2
    # i.e. the center of the image
    for i in range(padded_height):
      for j in range(padded_width):
        image_trans[i,j] = padded[i,j]*((-1)**(i+j))

    # Perform the fourier transformation
    fourier_transform = fftpack.fft2(image_trans)

    #filtrerer den fourier transformerte
    filtered = filtr*fourier_transform

    #transformerer med invers fourier
    filtered = fftpack.ifft2(filtered)

    #translerer tilbake
    filtered_translated = numpy.zeros(padded.shape)

    for i in range(padded_height):
        for j in range(padded_width):
            filtered_translated[i,j] = filtered[i,j]*((-1)**(i+j))
```

```
    #fjerner padding
    filtered_cropped = filtered_translated[0:height,0:width]

    return numpy.clip(filtered_cropped,0,1)
```

Her kan man velge forskjellige typer filtrering og størelsen på filtrene.

In [8]:

```
filterOptions = widgets.Dropdown(
    options=[
        'Box Blur','UnSharp using box blur','Butterworth High Pass', 'Butterworth Low Pass'
    ],
    description='Filter: ',
    disabled=False,
)

sizeOption = widgets.IntSlider(
    value = 15,
    min = 1,
    max = 100,
    description='Size: ',
    disabled=False,
    continuous_update = False
)



def filtrer(image, size, method):
    im = util.img_as_float(color.rgb2gray(io.imread(image)))

    height,width = im.shape

    if(method == 'Box Blur'):
        sizeOption.description = 'Size: '
        mask = numpy.ones((size,size))
        mask = mask / numpy.sum(mask)
        im = ndimage.convolve(im,mask,mode='constant', cval=0)
    elif(method == 'UnSharp using box blur'):
        sizeOption.description = 'Size: '
        mask = numpy.ones((size,size))
        mask = mask / numpy.sum(mask)
        imBlurred = ndimage.convolve(im,mask,mode='constant', cval=0)
        imMask = im - imBlurred
        im = im + imMask
    elif(method == 'Butterworth High Pass'):
        sizeOption.description = 'Cutoff%: '
        size = (size/100) * (width/2)
        filtr = butterworth(height*2,width*2,size,2.25)
        im = fourrierFilter(im,filtr)
    elif(method == 'Butterworth Low Pass'):
        sizeOption.description = 'Cutoff%: '
        size = (size/100) * (width/2)
        filtr = butterworthLowPass(height*2,width*2,size,2.25)
        im = fourrierFilter(im,filtr)
    elif(method == 'Unsharp using butterworth'):
        sizeOption.description = 'Cutoff%: '
        size = (size/100) * (width/2)
        filtr = butterworthLowPass(height*2,width*2,size,2.25)
        imBlurred = fourrierFilter(im,filtr)
        imMask = im - imBlurred
        im = im + imMask



    io.imshow(numpy.clip(im,0,1), cmap='gray')
    io.show()
```

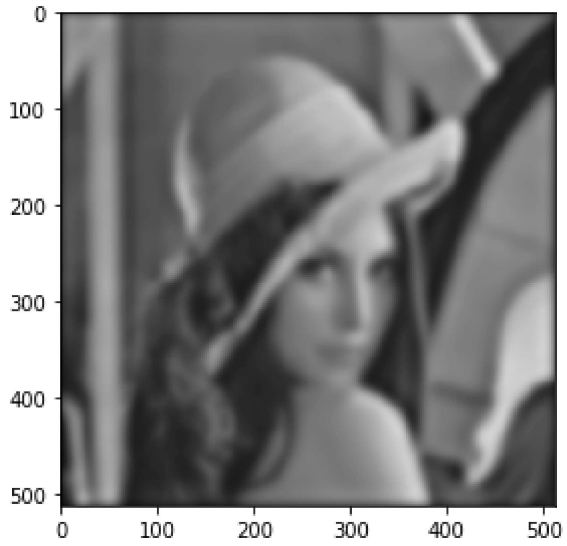
```
interactive(filtrer, image=inputImage, size=sizeOption, method=filterOptions)
```

Bilde: Lena

Cutoff%: 16

Filter: Butterworth Low Pass

```
C:\Users\Jim-Alexander\Anaconda3\lib\site-packages\ipykernel_launcher.py:5
8: ComplexWarning: Casting complex values to real discards the imaginary pa
rt
```



For å konkludere med noe her så må jeg si at jeg kan ikke se noen særlig forskjell på box filtret i spatial domet eller butterworth filtret i frekvensdomenet. Man har noe mer finkontroll over hvordan utjevningen skal bli med butterworth, men siden selve kalkuleringen tar mye lenger tid ville jeg heller gått for ett enkelt box filter. Selvfølgelig om jeg var en grafisk designer eller lignende hadde nok den ekstra finkontrollen kunnet gitt meg noe.