```
import numpy as np
from math import pi
from numpy.fft import fft2 as fft
from numpy.fft import fftshift as fftshift
from scipy.fft import fftfreq

# ======================================================================

def propagation(phi,wavelength, x,y,z,Z):
    """
    Propagates the wave function phi from z to Z.
    According to the Huygens-Fresnel principle, the computation is based on a
        superposition of spherical waves of a specified wavelength (paraxial
        approximation is used).
    Converts the coordinates x, y in the z plane to coordinates X, Y in Z plane.
    Returns the propagated wave function Phi and the new X and Y coordinates.
    """

    # No propagation
    if z == Z:
        Phi = phi
        X = x
        Y = y

    # Propagation
    else:
        # new coordinates
        yN,xN = np.shape(x)             # number of points
        xT = x[0,1]-x[0,0]              # spacing of the points
        yT = y[1,0]-y[0,0]
        xf = fftshift(fftfreq(xN,xT))   # Fourier space coordinates
        yf = fftshift(fftfreq(yN,yT))
        X = xf * wavelength * (Z-z)     # convert into real space coordinates
        Y = yf * wavelength * (Z-z)
        X,Y = np.meshgrid(X,Y)

        # Fourier transform-based propagation
        fourier = fftshift(fft(fftshift(\
                phi * np.exp( 1j * 2*pi/wavelength * (x**2 + y**2)/(Z-z)/2 ))))
        Phi = fourier * np.exp( 1j * 2*pi/wavelength * (X**2 + Y**2)/(Z-z)/2 )

    return Phi,X,Y

# ======================================================================
```

```
def lens(phi,wavelength,x,y,f):
    """
    Changes the shape of the wavefront  of the wave function phi.
    Takes the wave function wavelength and the coordinates x and y in the
        lens plane as input.
    Focuses as a perfect lens of focal length f.
    """

    # wavefront modulation by a phase shift
    Phi = phi * np.exp( - 1j * 2*pi/wavelength * (x**2 + y**2)/f/2 )
    return Phi

# ============================================================================

def aperture(phi,X,Y,shape,parameters,offset):
    """
    Limits the wave function phi in transversal direction.
    Takes the coordinates X and Y in the aperture plane as input.
    The shape of the aperture can be chosen out of followings:
        "c" = circular parameters = diameter
        "s" = square; parameters = side length
        "r" = rectangular; parameters = side lengths
    An offset of the aperture from the center of the computation area can be
        added.
    Returns the wave function Phi right behind the aperture.
    """

    # coordinates centering around the aperture
    x = X - offset[0]
    y = Y - offset[1]
    for i in range(len(parameters)):
        parameters[i] /=2

    # binary function to describe the aperture function
    if shape == "c":
        binary = 1 - np.heaviside( np.sqrt(x**2 + y**2) - parameters, 0 )
    elif shape == "s":
        binary = (1-np.heaviside( abs(x)-parameters,0 ))\
               * (1-np.heaviside( abs(y)-parameters, 0 ))
    elif shape == "r":
        binary = (1-np.heaviside( abs(x)-parameters[0],0 )) \
               * (1-np.heaviside( abs(y)-parameters[1], 0 ))

    # new wave function
    Phi = phi * binary
    return Phi
```

```
# ============================================================================

def phaseplate(phi,x,y,shape,parameters,offset,phase, absorption):
    """
    Induces a phase shift to the original wave function phi at the area of the
        phase plate.
    Takes the coordinates x and y in the aperture plane as input.
    The shape of the phase plate can be chosen out of followings:
    shapes:
        "c" = circular parameters = diameter
        "s" = square; parameters = side length
        "r" = rectangular; parameters = side lengths
    An offset of the phase plate from the center of the computation area can be
        added.
    The induced phase shift and absorption coefficient of the phase plate are
        to be set.
    """

    # coordinates centering around the aperture
    x = x - offset[0]
    y = y - offset[1]
    for i in range(len(parameters)):
        parameters[i] /=2

    # binary function to describe the area of the phase plate
    if shape == "c":
        binary = 1 - np.heaviside( np.sqrt(x**2 + y**2) - parameters, 0 )
    elif shape == "s":
        binary = (1-np.heaviside( abs(x)-parameters,0 ))\
               * (1-np.heaviside( abs(y)-parameters, 0 ))
    elif shape == "r":
        binary = (1-np.heaviside( abs(x)-parameters[0],0 )) \
               * (1-np.heaviside( abs(y)-parameters[1], 0 ))

    # inducing phase shift and limitting the amplitude ot the wave function
    Phi = phi * (1-absorption*binary* np.exp( 1j* phase*binary))
    return Phi

# ============================================================================
```

```
def grid(phi,x,y,spacing,aperture, absorption):
    """
    A cross grating-like pattern acting on the wave function phi.
    Takes the coordinates x and y in the grid plane as input.
    The spacing of the crosses and the size of the cros-free area (aperture)
        within the pattern are to be set.
    The absorption of the crosses can be modified.
    """

    xy = [x,y]
    Phi = phi
    for i in range(2):
        # when a grating is present in the direction
        if spacing[i] !=0:
            # transmission function of the grid
            grid = 1 - absorption\
                 * np.heaviside( abs( (xy[i] % spacing[i])) - aperture[i] , 0)
            # modulation of the wave function amplitude
            Phi *= grid
    return Phi

# ============================================================================

def checkerboard(phi,x,y,spacing,absorption):
    """
    A checkerboard-like pattern acting on the wave function phi.
    Takes the coordinates x and y in the grid plane as input.
    The spacing of the checkerboard pattern is to be set.
    The absorption of the checkerboard can be modified.
    """

    # transmission function of the checkerboard
    checkerboard = abs((-1) ** np.around(2*x/spacing[0]) \
                     + (-1) ** np.around(2*y/spacing[1])  )/2
    # modulation of the wave function amplitude
    Phi = phi * checkerboard
    return Phi

# ============================================================================
```

```
def diffracted(phi,x,y,wavelength, spacing):
    """
    Diffraction of the wave function phi on a squared grid and propagation of
        the four first diffraction maxima.
    Takes the wave function wavelength and the coordinates x and y in the grid
        plane as input.
    The spacing of the checkerboard pattern is to be set.
    """
    # superposition of the wave function propagating along the diffraction maxima
    Phi = phi * [ np.exp( 1j * (2*pi/wavelength) * x / spacing) \
                + np.exp(-1j * (2*pi/wavelength) * x / spacing) \
                + np.exp( 1j * (2*pi/wavelength) * y / spacing) \
                + np.exp(-1j * (2*pi/wavelength) * y / spacing) ]

    return Phi[0,:,:]
```