```
import numpy as np

def easy_mat(data, rows=2, cols=2):
    """ Makes it easy to create a matrix w/o typing so many []!
    Pass this function a tuple or list, optionally rows/columns

    Example:
    m = easy_mat([1, 2, 3, 4])

    """
    array = np.array(data)
    matrix = array.reshape(rows, cols)
    return(matrix)

def pretty_print(a):
    """ You don't really need to use this.
    However it makes a matrix look more like a matrix
    does in Matlab.

    'Pretty printing' is an old programming term that just means
    printing something in a more human friendly format.  It is left
    over from the dinosaur age when computer
    printouts were quite ugly by default.

    Example:
        m= ezmat([42, 3.14, 69, 12])
        pretty_print(m)
        Matrix follows:
            42.000    3.140
            69.000   12.000
    """
    print('Matrix follows:')
    if not isinstance(a,np.ndarray):
        print('WARNING: Pretty print will not work except on arrays.')
    for row in a:
            for col in row:
                print("{:8.3f}".format(col), end=" ")
            print("")


def trans(L):
    """
    L: translation distance (m)
    """
    T = easy_mat([1, L,  0, 1])
    return T


def thick(Rf, t, Rb, nE, nL):
    """
    Rf: radius of front surface (+ convex to left)
    t:  lens thickness (0 for thin lens approx)
    Rb: radius of back surface (+ to left convex)
    nE: 1    n of environment
    nL: 1.5  n of lens material
"""
    R1 = easy_mat([1, 0, 1/Rf*(nE/nL-1), nE/nL])
    T1 = easy_mat([1, t,  0, 1])
    R2 = easy_mat([1, 0, 1/Rb*(nL/nE-1), nL/nE])
    S = R2@T1@R1
    return S

def compose_thick(objct, image):
    """
    This is the part that you fill out.  Replace all the question marks
    appropriately.

    objct: The distance (in meters) to the object.  A positive
```

```
        number means object is to left of first lens.
    image: The distance (in meters) to the image.  This is something
    you have to guess.  Keep changing it until the
    absolute value of "B" in the matrix is <0.001.
    When it is, then "image" will be the image distance.
    T: for translation
    R: for refraction
    S: for "System matrix"
    """

    pass
# pass means do not do anything
# This is the function you have to write.  It's like
# compose_thin, but just will have more matrices


def compose_thin(objct, image):
    """
    I am doing this one for you so that you start out with
    working code.  You dont need this function for problem 2-6.
    Use compose_thick instead.
    T: for translation
    R: for refraction
    S: for "System matrix"
    """
    T1 = trans(L=objct)
    R1 = thick(Rf=0.06,t=0,Rb=-0.06,nE=1.5,nL=2.5)
    T2 = trans(L=image)
    S = T2@R1@T1
    return S


def main():
    S1 = compose_thin(0.1, 0.2)
    pretty_print(S1)
    S2 = compose_thin(0.1, 0.0815)
    pretty_print(S2)
    S3 = compose_thin(0.07, 0.126)
    pretty_print(S3)


if __name__ == '__main__':
    main()
```