```
from common import *


def b1(M):
    #Given:
    #   a matrix M
    #Return:
    #   the matrix S such that S[i,j] = M[i,j]*10+100
    #Hint: Trust that numpy will do the right thing
    S = M*10 + 100

    return S

def b2(t):
    #Given:
    #   a nxn matrix M1
    #   a nxn matrix M2
    #Return:
    #   the matrix P such that P[i,j] = M1[i,j]+M2[i,j]*10
    #Hint: Trust that numpy will do the right thing
    M1, M2 = t #unpack
    P = M1 + M2*10

    return P

def b3(t):
    #Given:
    #   a nxn matrix M1
    #   a nxn matrix M2
    #Return:
    #   the matrix P such that P[i,j] = M1[i,j]*M2[i,j]-10
    #Hint: By analogy to + , * will do the same thing
    M1, M2 = t #unpack
    P = M1*M2-10

    return P

def b4(t):
    #Given:
    #   a nxn matrix M1
    #   a nxn matrix M2
    #Return:
    #   the matrix product M1 M2
    #Hint: Not the same as * !
    M1, M2 = t #unpack
    P = M1.dot(M2)

    return P

def b5(M):
    #Given:
    #   a nxn matrix M of floats
    #Return:
    #   a nxn matrix M of integers
    #Hint: astype
    # M.astype(int)
    M=np.int64(M)

    return M

def b6(t):
    #Given:
    #   a nx1 vector M of integers
    #   a nx1 vector D of integers
```

```python
    #Return:
    #   the ratio (M/D), treating them as floats (i.e., 1/5 => 0.2)
    #Hint: dividing one integer by another is not the same as dividing two floats
    M, D = t #unpack
    P=M/D

    return P

def b7(M):
    #Given:
    #   a nxm matrix M
    #Return:
    #   a vector v of size (nxm)x1 containing the entries of M, listed in row order
    #Hint:
    #   1) np.reshape
    #   2) you can specify an unknown dimension as -1
    P= np.reshape(M,(-1,1))

    return P

def b8(n):
    #Given:
    #   an integer n
    #Return:
    #   a nx(2n) matrix of ones
    #Hint:
    #   data type not understood with calling np.zeros/np.ones is guaranteed
    #   to be an issue where you passed in two arguments, not a tuple
    P = np.ones((n,2*n), dtype=float)

    return P

def b9(M):
    #Given:
    #   a matrix M where each entry is between 0 and 1
    #Return:
    #   a matrix S where S[i,j] = True if M[i,j] > 0.5
    #Hint: Trust python to do the right thing
    S = M > 0.5

    return S

def b10(n):
    #Given:
    #   an integer n
    #Return:
    #   the n-entry vector of 0, ..., n-1
    #Hint: range+np.array/np.arange
    result=np.arange(n)

    return result

def b11(t):
    #Given:
    #   a NxF matrix A
    #   a Fx1 vector v
    #Return:
    #   the matrix-vector product Av
    A, v = t
    P = A.dot(v)

    return P

def b12(t):
    #Given:
```

```
    #   a NxN matrix A, full rank
    #   a Nx1 vector v
    #Return:
    #   the inverse of A times v: A^-1 v
    A, v = t
    P = np.linalg.inv(A).dot(v)

    return P


def b13(t):
    #Given:
    #   a Nx1 vector u
    #   a Nx1 vector v
    #Return:
    #   the innner product u^T v
    #Hint:
    #   .T
    u, v = t
    P=np.transpose(u).dot(v)

    return P

def b14(v):
    #Given:
    #   a Nx1 vector v
    #Return:
    #   the L2-norm without calling np.linalg.norm
    #norm = (\sum_i=1^N v[i]**2)**0.5
    P = np.linalg.norm(v)

    return P

def b15(t):
    #Given:
    #   a NxF matrix M
    #   an integer i
    #Return:
    #   the ith row of M
    M, i = t
    P = M[i,:]

    return P

def b16(M):
    #Given:
    #   a NxF matrix M
    #Return:
    #   the sum of all the entrices of the matrix
    #Hint:
    #   np.sum
    P = np.sum(M)

    return P

def b17(M):
    #Given:
    #   a NxF matrix M
    #Return:
    #   a N-entry vector S where S[i] is the sum along row i of M
    #Hint:
    #   np.sum has an axis optional arg; note keepdims if you already know this
    P = np.sum(M, axis=1)

    return P
```

```

def b18(M):
    #Given:
    #   a NxF matrix M
    #Return:
    #   a F-entry vector S where S[j] is the sum along column j of M
    #Hint: same as above
    P = np.sum(M, axis=0)

    return P

def b19(M):
    #Given:
    #   a NxF matrix M
    #Return:
    #   a Nx1 matrix S where S[i,1] is the sum along row i of M
    #Hint:
    #   Watch axis, keepdims
    P = np.sum(M, axis=1)
    P = P.reshape(-1,1)

    return P


def b20(M):
    #Given:
    #   a NxF matrix M
    #Return:
    #   a Nx1 matrix S where S[i] is the L2-norm of row i of M
    #Hint:
    #   Put it together
    P =np.linalg.norm(M,axis=1)
    P = P.reshape(-1,1)

    return P
```