

```

1  from common import *
2
3
4  def b1(M):
5      #Given:
6      #    a matrix M
7      #Return:
8      #    the matrix S such that  $S[i,j] = M[i,j]*10+100$ 
9      #Hint: Trust that numpy will do the right thing
10     S = M*10 + 100
11
12     return S
13
14  def b2(t):
15      #Given:
16      #    a nxn matrix M1
17      #    a nxn matrix M2
18      #Return:
19      #    the matrix P such that  $P[i,j] = M1[i,j]+M2[i,j]*10$ 
20      #Hint: Trust that numpy will do the right thing
21      M1, M2 = t #unpack
22      P = M1 + M2*10
23
24      return P
25
26  def b3(t):
27      #Given:
28      #    a nxn matrix M1
29      #    a nxn matrix M2
30      #Return:
31      #    the matrix P such that  $P[i,j] = M1[i,j]*M2[i,j]-10$ 
32      #Hint: By analogy to + , * will do the same thing
33      M1, M2 = t #unpack
34      P = M1*M2-10
35
36      return P
37
38  def b4(t):
39      #Given:
40      #    a nxn matrix M1
41      #    a nxn matrix M2
42      #Return:
43      #    the matrix product M1 M2
44      #Hint: Not the same as * !
45      M1, M2 = t #unpack
46      P = M1.dot(M2)
47
48      return P
49
50  def b5(M):
51      #Given:
52      #    a nxn matrix M of floats
53      #Return:
54      #    a nxn matrix M of integers
55      #Hint: astype
56      # M.astype(int)
57      M=np.int64(M)
58
59      return M
60
61  def b6(t):
62      #Given:
63      #    a nx1 vector M of integers
64      #    a nx1 vector D of integers

```

```

65     #Return:
66     # the ratio (M/D), treating them as floats (i.e., 1/5 => 0.2)
67     #Hint: dividing one integer by another is not the same as dividing two floats
68     M, D = t #unpack
69     P=M/D
70
71     return P
72
73 def b7(M):
74     #Given:
75     # a nxm matrix M
76     #Return:
77     # a vector v of size (nxm)x1 containing the entries of M, listed in row order
78     #Hint:
79     # 1) np.reshape
80     # 2) you can specify an unknown dimension as -1
81     P= np.reshape(M,(-1,1))
82
83     return P
84
85 def b8(n):
86     #Given:
87     # an integer n
88     #Return:
89     # a nx(2n) matrix of ones
90     #Hint:
91     # data type not understood with calling np.zeros/np.ones is guaranteed
92     # to be an issue where you passed in two arguments, not a tuple
93     P = np.ones((n,2*n), dtype=float)
94
95     return P
96
97 def b9(M):
98     #Given:
99     # a matrix M where each entry is between 0 and 1
100    #Return:
101    # a matrix S where S[i,j] = True if M[i,j] > 0.5
102    #Hint: Trust python to do the right thing
103    S = M > 0.5
104
105    return S
106
107 def b10(n):
108     #Given:
109     # an integer n
110     #Return:
111     # the n-entry vector of 0, ..., n-1
112     #Hint: range+np.array/np.arange
113     result=np.arange(n)
114
115     return result
116
117 def b11(t):
118     #Given:
119     # a NxF matrix A
120     # a Fx1 vector v
121     #Return:
122     # the matrix-vector product Av
123     A, v = t
124     P = A.dot(v)
125
126     return P
127
128 def b12(t):
129     #Given:

```

```

130     # a NxN matrix A, full rank
131     # a Nx1 vector v
132     #Return:
133     # the inverse of A times v:  $A^{-1} v$ 
134     A, v = t
135     P = np.linalg.inv(A).dot(v)
136
137     return P
138
139
140 def b13(t):
141     #Given:
142     # a Nx1 vector u
143     # a Nx1 vector v
144     #Return:
145     # the innner product  $u^T v$ 
146     #Hint:
147     # .T
148     u, v = t
149     P=np.transpose(u).dot(v)
150
151     return P
152
153 def b14(v):
154     #Given:
155     # a Nx1 vector v
156     #Return:
157     # the L2-norm without calling np.linalg.norm
158     #norm = ( $\sum_{i=1}^N v[i]**2$ )**0.5
159     P = np.linalg.norm(v)
160
161     return P
162
163 def b15(t):
164     #Given:
165     # a NxF matrix M
166     # an integer i
167     #Return:
168     # the ith row of M
169     M, i = t
170     P = M[i,:]
171
172     return P
173
174 def b16(M):
175     #Given:
176     # a NxF matrix M
177     #Return:
178     # the sum of all the entrices of the matrix
179     #Hint:
180     # np.sum
181     P = np.sum(M)
182
183     return P
184
185 def b17(M):
186     #Given:
187     # a NxF matrix M
188     #Return:
189     # a N-entry vector S where  $S[i]$  is the sum along row i of M
190     #Hint:
191     # np.sum has an axis optional arg; note keepdims if you already know this
192     P = np.sum(M, axis=1)
193
194     return P

```

```

195
196 def b18(M):
197     #Given:
198     #    a NxF matrix M
199     #Return:
200     #    a F-entry vector S where S[j] is the sum along column j of M
201     #Hint: same as above
202     P = np.sum(M, axis=0)
203
204     return P
205
206 def b19(M):
207     #Given:
208     #    a NxF matrix M
209     #Return:
210     #    a Nx1 matrix S where S[i,1] is the sum along row i of M
211     #Hint:
212     #    Watch axis, keepdims
213     P = np.sum(M, axis=1)
214     P = P.reshape(-1,1)
215
216     return P
217
218
219 def b20(M):
220     #Given:
221     #    a NxF matrix M
222     #Return:
223     #    a Nx1 matrix S where S[i] is the L2-norm of row i of M
224     #Hint:
225     #    Put it together
226     P = np.linalg.norm(M,axis=1)
227     P = P.reshape(-1,1)
228
229     return P

```