SELECTED SOLUTIONS

CHAPTER 4

Exercise Set 4.1.1

1. A. $a_{1,1} = 7$

   B. $a_{2,3} = 0$

2. A. 2 x 3

   B. 2 x 2

   C. 3 x 1

   D. 1 x 3

   E. 3 x 3

3. A. $\begin{bmatrix} 1 & 3 & 11 \\ 5 & 15 & 5 \end{bmatrix}$

   B. $\begin{bmatrix} 6 & -20 \\ 25 & -6 \end{bmatrix}$

Exercise Set 4.1.2

1. Modify Program 4.1.1 so that A = $\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$.

```
A = [[ 1, 2],[3,4], [5, 6]]
r = int(input("row? "))
c = int(input("column? "))
if r in [0,1] and c in [0,1,2]:
   print(A[r][c])
else:
   print("out of range")
```

2. Modify Program 4.1.2 so that instead of assigning 0's to each matrix element, it assigns a random number between 1 and 10 to each element.

```
from random import randint
r = int(input("Rows? "))
```

```
c = int(input("Columns? "))
A = [ ]                              # initialize the matrix
for i in range(r):                   # for each row
   row = [ ]                         # initialize row i
   for j in range(c):                # for each column
      row.append(randint(1,10))      # append random integer to row i
   A.append(row)                     # append row to matrix A

print(A)
```

3. Write a program that calculates and output the sum of the following two matrices:

$$A = \begin{bmatrix} 1 & 0 \\ -2 & 5 \end{bmatrix} \quad B = \begin{bmatrix} -3 & 10 \\ 4 & -7 \end{bmatrix}$$

```
A = [[1,0], [-2,5]]
B = [[-3,10], [4,-7]]
C = [[0,0], [0,0]]
for i in range(2):
   for j in range(2):
      C[i][j] = A[i][j]+B[i][j]
print(C)
```

4. The result would be:
[[1, 0], [-2, 5], [-3, 10], [4, -7]]

which is $\begin{bmatrix} 1 & 0 \\ -2 & 5 \\ -3 & 10 \\ 4 & -7 \end{bmatrix}$

so rather than perform matrix addition, the effect of A+B is to append matrix B to the bottom of matrix A.

Exercise Set 4.1.3

1. listA = [5*i **for** i **in range**(10))]

2. listB = [i**2 **for** i **in** listC]

3. listC = [i*j **for** i **in** list D **for** j **in** listE]

Exercise Set 4.1.4

1. Write a brief funtion called init_matrix(r,c) that returns a matrix M with dimensions r x c in which all the entries are 0.

```
def init_matrix(r,c):
    M = [ [ 0 for j in range(c) ] for i in range(r) ]
    return M
```

Exercise Set 4.1.5

1. A. $\begin{bmatrix} 2 & 10 \\ 0 & -12 \end{bmatrix}$

   B. $\begin{bmatrix} 2 & 4/3 \\ -3 & -1/3 \end{bmatrix}$

   C. $\begin{bmatrix} -13 & 3 \\ -11 & 19 \end{bmatrix}$

   D. $\begin{bmatrix} 24 & 3 \\ 24 & 3 \end{bmatrix}$

2. No, you would end up with:

[[2, 4], [6, 8], [2, 4], [6, 8]]

instead of:

[[4, 8], [12, 16]]

3. Write a program that squares all the entries of $\begin{bmatrix} 11 & 12 & 13 \\ 14 & 15 & 16 \end{bmatrix}$.

```
import matrix_fcn as mf

A = [[11,12,13],[14,15,16]]

r = len(A)                    # of rows
c = len(A[0])                  # of columns
B = mf.init_matrix(r,c)

for i in range(r):
    B[i] = [A[i][j]**2 for j in range(c)]

mf.print_matrix(B)
```

4. Write a program that performs the matrix operations:

$$3\begin{bmatrix} 1 & 5 \\ -1 & -5 \end{bmatrix} + 4\begin{bmatrix} -4 & -3 \\ -.2 & -1 \end{bmatrix}$$

```
import matrix_fcn as mf

A = [[1,5],[-1,-5]]
B = [[-4, -3],[-2, -1]]
r = len(A)                          # of rows
c = len(A[0])                        # of columns
C = mf.init_matrix(r,c)

for i in range(r):
   C[i] = [3*A[i][j]+4*B[i][j] for j in range(c)]

mf.print_matrix(C)
```

5. Write a program that generates a multiplication table for integers up to 9.

```
import matrix_fcn as mf

# row 0 (headers)
A = mf.init_matrix(10,10)
A[0] = [i for i in range(10)]

# rows 1 - 9:
for i in range(1,10):
   A[i][0]=i                  # 1st column is row number (multiplier)
   for j in range(1,10):     # columns 1 - 9
      A[i][j] = i*j

mf.print_matrix(A)
```

Exercise Set 4.1.6

1. Modify function input_matrix() so that it takes parameters r and c.

```
def input_matrix(r,c):
   if r == 0:
      r = int(input("# of rows? "))
   if c == 0:
      c = int(input("# of columns? "))
:
:
#remainder of function
```

2. Modify program **scalar_mult** so that the user is prompted to enter a scalar, the dimensions of a matrix, and the matrix. Perform scalar multiplication.

```
import matrix_fcn as mf
scalar = int(input("scalar? "))
A = mf.input_matrix(0,0)
r = len(A)
c = len(A[0])
B = mf.init_matrix(r,c)
for i in range(r):
   B[i] = [scalar*A[i][j] for j in range(c)]
mf.print_matrix(B)
```

4. Write a program that asks the user for a single number N. Create a square matrix with N rows and N columns. Each element of the matrix will be 0 EXCEPT for the elements along the diagonal, which will be 1. For example, a 3x3 matrix would look like this:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

```
import matrix_fcn as mf
N = int(input("size of square matrix? "))
A = mf.init_matrix(N,N)
for i in range(N):
   A[i][i]=1
mf.print_matrix(A)
```

Exercise Set 4.1.7

1. Fill in the yellow highlighted code to generate each row of the lotto matrix.

```
from random import randint
import matrix_fcn as mf

lotto = mf.init_matrix(6,6)
for i in range(6):
   for j in range(5):
      lotto[i][j] = randint(1,47)
   lotto[i][5] = randint(30,50)
mf.print_matrix(lotto)
```

Exercise Set 4.1.8

1. Incorporate the use of **randint_norepeat()** into program lottery.

```
from random import randint
import matrix_fcn as mf

def randint_norepeat(m,k,n):
   # returns a list of m unique random numbers in the range [k,n]

   from random import shuffle
   numbers = list(range(k,n+1))
   shuffle(numbers)
   random_list = [ ]
   for i in range(m):
      random_list.append(numbers.pop())
   return random_list

lotto = []
last_col = randint_norepeat(5,1,27)
for i in range(5):

   row_list = randint_norepeat(5,1,47)
   row_list.append(last_col.pop())
   lotto.append(row_list)

mf.print_matrix(lotto)
```

2. Modify program so that the first 5 numbers in each row are output sequentially lowest to highest.

Insert after row_list = randint_norepeat(5,1,47):

```
row_list.sort()
```

Exercise Set 4.2.1

1. 7

2. not possible

3. 31

4. -53

5. not possible

6. 146

Exercise Set 4.2.2

1. Modify program to accept and check user input.

```
import poly_fcn as pf

A_str = input("A? ")
A = [int(a) for a in A_str.split(',')]
B_str = input("B? ")
B = [int(b) for b in B_str.split(',')]


if len(A) != len(B):
   print("Sorry, A and B must be the same size.")
else:
   result = 0
   for k in range(len(A)):
      result +=A[k]*B[k]
   print("Product is",result)
```

Exercise Set 4.2.3

1. $\begin{bmatrix} 58 & 60 \\ -14 & -16 \end{bmatrix}$
2. $\begin{bmatrix} 28 & 20 \\ 24 & 14 \end{bmatrix}$
3. $\begin{bmatrix} 69 & 68 \\ 67 & -51 \end{bmatrix}$
4. $\begin{bmatrix} 14 & 11 & 30 \\ 86 & 79 & 120 \\ 2 & 11 & -75 \end{bmatrix}$
5. $\begin{bmatrix} 1 & 11 & -8 \\ 7 & 7 & 2 \\ 1 & 10 & -4 \end{bmatrix}$
6. No.
7. Multiplying by the identity matrix results in the same matrix.
8. $\begin{bmatrix} 84 & 72 & 75 & 91 \\ 82 & 68 & 74 & 85 \\ 99 & 85 & 86 & 95 \\ 60 & 74 & 68 & 75 \\ 89 & 91 & 96 & 0 \end{bmatrix} \begin{bmatrix} .15 \\ .3 \\ .4 \\ .15 \end{bmatrix}$

   Amelia 77.85%, Bradley 75.05%, Kimiko 89%, Wesley 69.65%, Antwone 79.05%

Exercise Set 4.2.4

1. Fill in yellow highlighted code to calculate value of $p_{i,j}$.

```
import matrix_fcn as mf

A = [[1,2,3],[4,5,6]]
B = [[7,8],[9,10],[11,12]]
A_num_rows = len(A)
A_num_cols = len(A[0])
B_num_rows = len(B)
B_num_cols = len(B[0])

if A_num_cols != B_num_rows:
   print("Sorry, # of A columns must equal # of B rows")
else:
   P = mf.init_matrix(A_num_rows,B_num_cols)
   for i in range(A_num_rows)        # for each row in A
      for j in range(B_num_cols)     # for each column in B
            for k in range(A_num_cols):
               P[i][j] += A[i][k]*B[k][j]
   mf.print_matrix(P)
```

2. Modify the program so that the user can input any two compatible matrices A and B.

Replace the assignment statements for A and B with:

```
print("Matrix A:")
A = mf.input_matrix(0,0)
print("Matrix B:")
B = mf.input_matrix(0,0)
```

3. Convert the program to a function.

```
def matrix_product(A,B):
   import matrix_fcn as mf

   A_num_rows = len(A)
   A_num_cols = len(A[0])
   B_num_rows = len(B)
   B_num_cols = len(B[0])

   if A_num_cols != B_num_rows:
```

```
    print("Sorry, # of A columns must equal # of B rows")
    return [[0]]
  else:
    P = mf.init_matrix(A_num_rows,B_num_cols)
    for i in range(A_num_rows):        # for each row in A
      for j in range(B_num_cols):      # for each column in B
          for k in range(A_num_cols):
            P[i][j] += A[i][k]*B[k][j]
  return P
```

Exercise Set 4.3.1

1. $\begin{bmatrix} 3/29 & -2/29 \\ 7/29 & 5/29 \end{bmatrix}$
2. $\begin{bmatrix} -5 & -2 \\ 7 & 3 \end{bmatrix}$
3. $\begin{bmatrix} -2 & -1 \\ -5 & -3 \end{bmatrix}$
4. noninvertible
5. $\begin{bmatrix} -2 & 1 \\ 3/2 & -1/2 \end{bmatrix}$
6. Write a program to calculate and output the inverse of a 2x2 matrix.

```
import matrix_fcn as mf

A = mf.input_matrix(2,2)

I = [[0,0],[0,0]]
d = A[0][0] *A[1][1]-A[0][1]*A[1][0]
if d==0:
   print("noninvertible")
else:
   I[0][0] = round((1/d)*A[1][1],2)
   I[0][1] = round( (1/d)*-1*A[0][1],2)
   I[1][0] = round( (1/d)*-1*A[1][0],2)
   I[1][1] = round( (1/d)*A[0][0],2)
   print(I)
```

Exercise Set 4.3.2

1. You get a ValueError in the **print_matrix** function because it is attempting to output a float number using an integer format.

2. You get very long float numbers in the answer.

3. Error message.

```
import matrix_fcn as mf

def matrix_inverse(A)

   I = [[0,0],[0,0]]
   d = A[0][0] *A[1][1]-A[0][1]*A[1][0]
   if d==0:
      print("noninvertible")
      return [[ 0 ]]
   else:
   I[0][0] = round((1/d)*A[1][1],2)
   I[0][1] = round( (1/d)*-1*A[0][1],2)
   I[1][0] = round( (1/d)*-1*A[1][0],2)
   I[1][1] = round( (1/d)*A[0][0],2)
   return I

# main program
A = mf.input_matrix(2,2)
if A != [[ 0 ]]:
   I = mf.matrix_inverse(A)
   print("\nInverse:")
   print(I)
```

4. The program outputs a 2x2 inverse based on $a_{1,1}$, $a_{1,2}$, $a_{2,1}$ and $a_{2,2}$.

```
import matrix_fcn as mf

def matrix_inverse(A)
   if len(A) != 2 or len(A[0]) !=2:
      print("A must be 2x2")
      return [[ 0 ]]
   else:
      I = [[0,0],[0,0]]
      d = A[0][0] *A[1][1]-A[0][1]*A[1][0]
      if d==0:
         print("noninvertible")
         return [[ 0 ]]
      else:
         I[0][0] = round((1/d)*A[1][1],2)
         I[0][1] = round( (1/d)*-1*A[0][1],2)
```

```
        I[1][0] = round( (1/d)*-1*A[1][0],2)
        I[1][1] = round( (1/d)*A[0][0],2)
        return I

# main program
:
:
```

Exercise Set 4.3.3

1. $\begin{bmatrix} -3 \\ 5 \end{bmatrix}$
2. $\begin{bmatrix} 2 \\ -1 \end{bmatrix}$
3. $\begin{bmatrix} 4/3 \\ 1/5 \end{bmatrix}$
4. $\begin{bmatrix} 1/2 \\ 2 \end{bmatrix}$

Exercise Set 4.3.4

1. The output is [[5.8800000000001], [-0.89999999999995]]

   No, the actual answer is [[6], [-1]]

   The reason this is inaccurate is because the **matrix_inverse** function is rounding the inverse entries to 2 decimal places. Fix this by removing the **round()** functions from the **matrix_inverse** function.

2. The output is:

```
Noninvertible
Sorry, A and B must be compatible.
[ [ 0 ] ]
```

```
import matrix_fcn as mf

print("matrix A:")
A = mf.input_matrix(2,2)
print("matrix B:")
B = mf.input_matrix(2,1)

I = mf.matrix_inverse(A)
if I = [ [ 0 ] ]:
```

```
    print("No solution")
else:
    X = mf.matrix_product(I,B)
    print(X)
```

3. You get a ValueError message because the input_matrix function is expecting integer inputs.

Exercise Set 4.4.1

1. $\begin{bmatrix} 3 & -4 \\ 1/2 & -3 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 4 \\ -1/2 \end{bmatrix}$ Answer: (x,y) = (2, ½)
2. $\begin{bmatrix} 3 & 2 \\ 1 & -7 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 2 \\ -30 \end{bmatrix}$ Answer: (-2, 4)
3. $\begin{bmatrix} 4 & 5 \\ 0 & -2 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} -3 \\ 8 \end{bmatrix}$ Answer: $(-\frac{23}{4}, 4)$
4. $\begin{bmatrix} 4 & 8 \\ 2 & 4 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 11 \\ 17 \end{bmatrix}$ No solution
5. $\begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 4 \\ 8 \end{bmatrix}$ Answer: (6, 2)
6. $\begin{bmatrix} 1 & 2 \\ 1 & \frac{1}{2} \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 4 \\ 4 \end{bmatrix}$ Answer: (4, 0)

Exercise Set 4.4.2

1. $\left(-\frac{8}{5}, \frac{7}{5}\right)$
2. (½, 2)
3. No solution. Inconsistent.
4. (6, -1)
5. No solution. Consistent-Dependent.
6. (u, v) = (5, 3) so (x, y) = $\left(\frac{1}{5}, \frac{1}{3}\right)$

Exercise Set 4.4.3

1. $\left(\frac{4}{3}, \frac{1}{5}\right)$
2. $\left(-\frac{5}{3}, 1\right)$
3. $\left(-\frac{1}{2}, 2\right)$
4. No solution.
5. (-2, 4)
6. 110 adults, 215 military.
7. 60 lb. dark chocolate, 40 lb. Ruby chocolate
8. $90,000 in AA bonds, $60,000 in bank CD's.

Exercise Set 4.5.1

Same as Exercise Set 4.4.3 #1-4.

Exercise Set 4.5.2

1. Modify program to accept input statements and use more efficient assignment statements.

```
from fractions import Fraction
import matrix_fcn as mf

print("Matrix A:")
A = mf.input_matrix(2,2)

print("Matrix B:")
B = mf.input_matrix(2,2)

a,b,c,d = A[0][0], A[0][1], A[1][0], A[1][1]
s,t = B[0][0], B[1][0]

detA = a*d-b*c
Ax = s*d-b*t
Ay = a*t-s*c

print("X=",Fraction(Ax,detA))
print("Y=",Fraction(Ay,detA))
```

2. Modify the program to print "no solution" as needed.

Insert the following lines above num_X = s*d – b*t:

```
if detA == 0:
   print("No solution")
else:
```

3. Bypass use of variables **a, b, c, d, s, t** entirely. Have assignment statements for detA, Ax and Ay use entries of A and B.

```
from fractions import Fraction
import matrix_fcn as mf

print("Matrix A:")
A = mf.input_matrix(2,2)

print("Matrix B:")
B = mf.input_matrix(2,1)

detA = A[0][0]*A[1][1]-A[0][1]*A[1][0]
Ax = B[0][0]*A[1][1]-A[0][1]*B[1][0]
Ay = A[0][0]*B[1][0]-B[0][0]*A[1][0]

print("X=",Fraction(Ax,detA))
print("Y=",Fraction(Ay,detA))
```

Exercise Set 4.5.3

1. $a_{3,1}\begin{bmatrix} a_{1,2} & a_{1.3} \\ a_{2.2} & a_{2,3} \end{bmatrix} - a_{3,2}\begin{bmatrix} a_{1,1} & a_{1,3} \\ a_{2,1} & a_{2,3} \end{bmatrix} + a_{3,3}\begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{bmatrix}$

2. 54

3. 0

4. Write a function that will return the determinant of a 3x3 matrix.

```
def determ(M):

   return M[0][0]*(M[1][1]*M[2][2]-M[1][2]*M[2][1])-M[0][1]*(M[1][0]*M[2][2]-

M[1][2]*M[2][0])+ M[0][2]*(M[1][0]*M[2][1]-M[1][1]*M[2][0])

# main program (for testing)
import matrix_fcn as mf

A = [[-5, 0, 1], [1, 2, -1], [-3, 4, 1]]
det = mf.determ(A)
```

```
print("Determinant of:")
mf.print_matrix(A)
print("\nis",det)
```

Exercise Set 4.5.4

1. (-3, 0, 4)

2. (3, -2, 1)

3. (1, 3, -2)

Exercise Set 4.5.5

1. Fill in the yellow highlighted code to calculate and output x, y, z.

```
print("(", Fraction(Dx/D), ",", Fraction(Dy/D), ",", Fraction(Dz/D), ")")
```

2. Modify program cramer3d as instructed.

```
from fractions import Fraction
from copy import deepcopy
import matrix_fcn as mf

print("Matrix A:")
A = mf.input_matrix(3,3)
B_str = input("Matrix B: ")
B = [float(k) for k in B_str.split(',')]

At = deepcopy(A)                    # At is a temporary clone of A
D = mf.determ(A)

      # Replace A col 1 with B
At[0][0], At[1][0], At[2][0] = B
Dx = mf.determ(At)
     # Replace A col 2 with B
At = deepcopy(A)
At[0][1], At[1][1],At[2][1] = B
Dy = mf.determ(At)
      # Replace A col 3 with B
At = deepcopy(A)
At[0][2], At[1][2], At[2][2] = B
Dz = mf.determ(At)

if D == 0:
   if Dx == 0 and Dy == 0 and Dz == 0:
```

```
      print("CONSISTENT DEPENDENT")
    else:
      print("INCONSISTENT")
else:
  print("(", Fraction(Dx/D), ",", Fraction(Dy/D), ",", Fraction(Dz/D), ")")
```

Exercise Set 4.6.1

1. $y = -x^2 + 4x - 3$

2. $y = -x^2 + 3x - 2$

3. $f(x) = -0.11x^2 + 1.43x - 0.67$; \$21.51

CHAPTER PROJECTS

III.

1. THE CROW FLIES AT MIDNIGHT