Adamson University
College of Engineering
Computer Engineering Department

Linear Algebra

Laboratory Activity No.

Matrices

Submitted by:
Guy, Lawrence Adrian B.

Instructor:
Engr. Dylan Josh D. Lopez

November 01, 2020

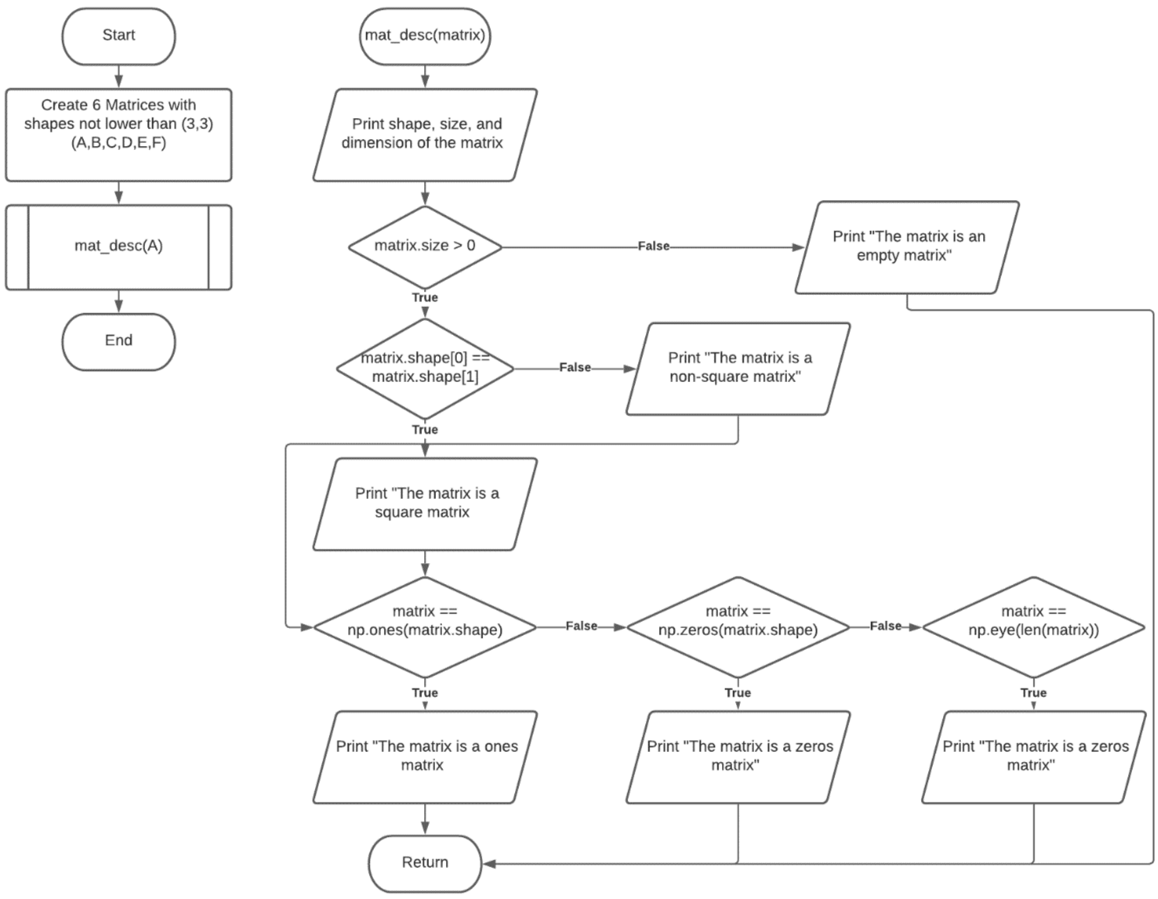
I. Objectives

The laboratory aims to be familiar with matrices and their relation to linear equations. The activity shows the declaration of matrices in Python and the different categories of matrices. The activity also shows basic matrix operations using Python.

II. Methods

The practices of the activity are performing basic matrix operations and programming of different matrix equations and operations in Python. The matrices in the activity are classified into two, shape and elemental values. The matrix operations included in the activity are addition, subtraction, and element-wise multiplication.

The first deliverable of the activity is to create a function named mat_desc(). The function describes the matrix's shape, size, and rank. The function also categorizes the matrix if it is a square or non-square, empty, identity, ones, or zeros matrix.



Start
Create 6 Matrices with shapes not lower than (3,3) (A,B,C,D,E,F)
mat_desc(A)
End
mat_desc(matrix)
Print shape, size, and dimension of the matrix
matrix.size > 0
False
Print "The matrix is an empty matrix"
True
matrix.shape[0] == matrix.shape[1]
False
Print "The matrix is a non-square matrix"
True
Print "The matrix is a square matrix
matrix == np.ones(matrix.shape)
False
matrix == np.zeros(matrix.shape)
False
matrix == np.eye(len(matrix))
True
True
True
Print "The matrix is a ones matrix
Print "The matrix is a zeros matrix"
Print "The matrix is a zeros matrix"
Return

Figure 1 Flowchart for task 1

The mat_desc() function has a parameter named matrix. The function uses np.ndarray.shape, np.ndarray.size, and np.ndarray.ndim to describe the shape, size, and dimensions or rank of the matrix [1][2][3]. Next, the function checks whether the matrix size is greater than 0 if it isn't then it will print that the matrix is an empty matrix then return. If it is, then it will check if the rows and columns of the matrix are equal by using np.ndarray.shape. If the shape of the rows and columns are equal, it will print that the matrix is a square matrix. If the shape of the rows is not equal then it will print that the matrix is a non-square matrix. Next, the function will check if the matrix is an identity, zeros, or ones matrix using np.eye(), np.zeros(), and np.ones(). The function np.eye() returns an array with zeros and ones, wherein the ones are diagonally aligned [4]. The function np.zeros() returns an array with all zeros as elements [5]. The function np.ones() returns an array with all ones as elements [6]. The function will do this by using an if-else statement if the matrix is equal to np.ones(matrix.shape) then it will print that the matrix is a ones matrix. If it is not then it will check if the matrix is equal to the np.zeros(matrix.shape). If it is then it will print that the matrix is a zeros matrix. Lastly, the function will check if the matrix is equal to the np.eye(len(matrix)). If it is then it will print that the matrix is an identity matrix. If the parameter isn't a matrix then the function will output an error.

The second deliverable is to create a function named mat_operations(). The function determines if the first parameter and the second parameter is a matrix or a scalar. Then the function will describe the matrix if the parameter is a matrix. Lastly, the function will perform matrix operations on the first and second parameters.

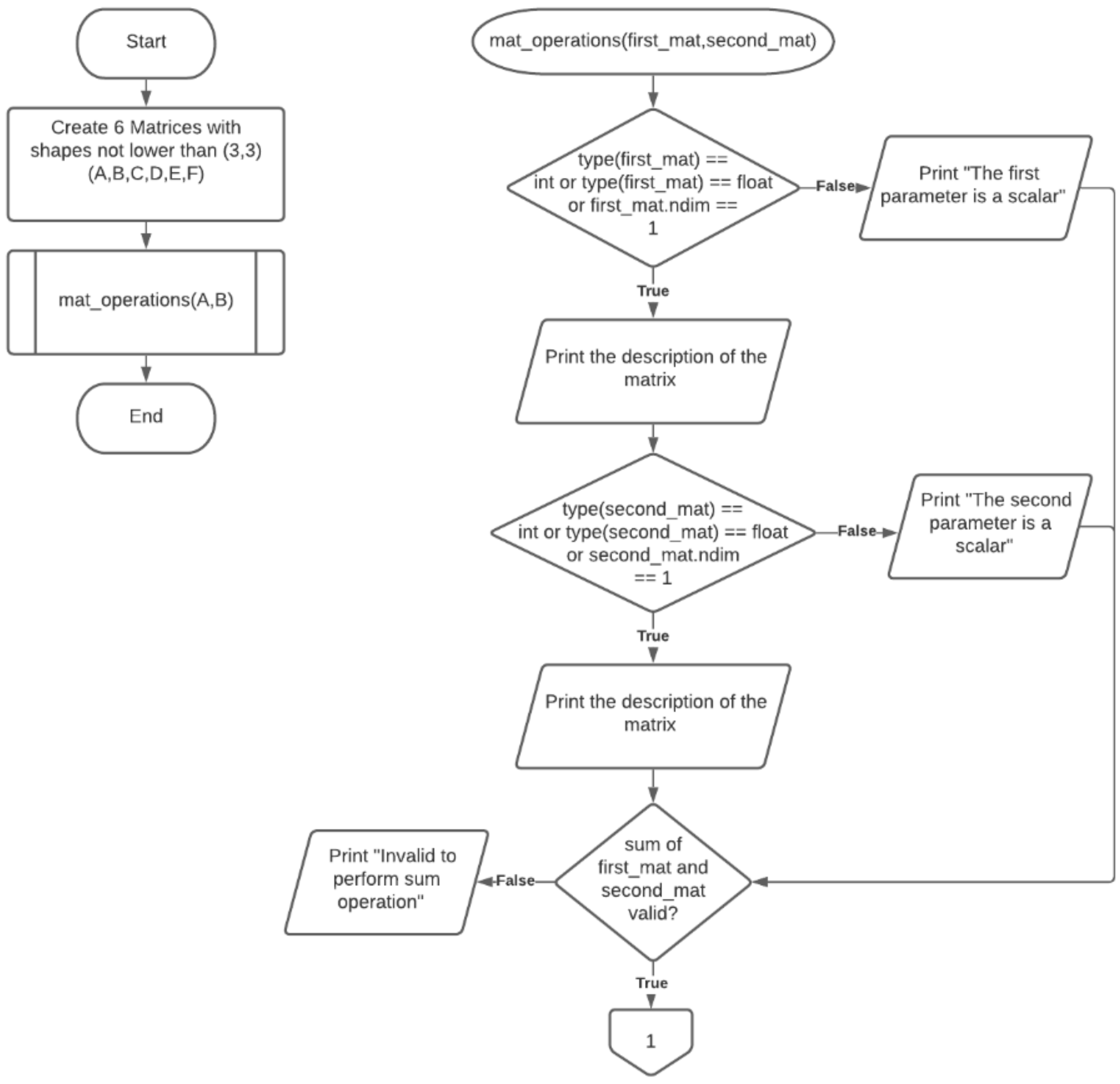

Start
Create 6 Matrices with shapes not lower than (3,3) (A,B,C,D,E,F)
mat_operations(A,B)
End
mat_operations(first_mat,second_mat)
type(first_mat) == int or type(first_mat) == float or first_mat.ndim == 1
False
Print "The first parameter is a scalar"
True
Print the description of the matrix
type(second_mat) == int or type(second_mat) == float or second_mat.ndim == 1
False
Print "The second parameter is a scalar"
True
Print the description of the matrix
sum of first_mat and second_mat valid?
False
Print "Invalid to perform sum operation"
True
1

Figure 2.1 Flowchart for task 2

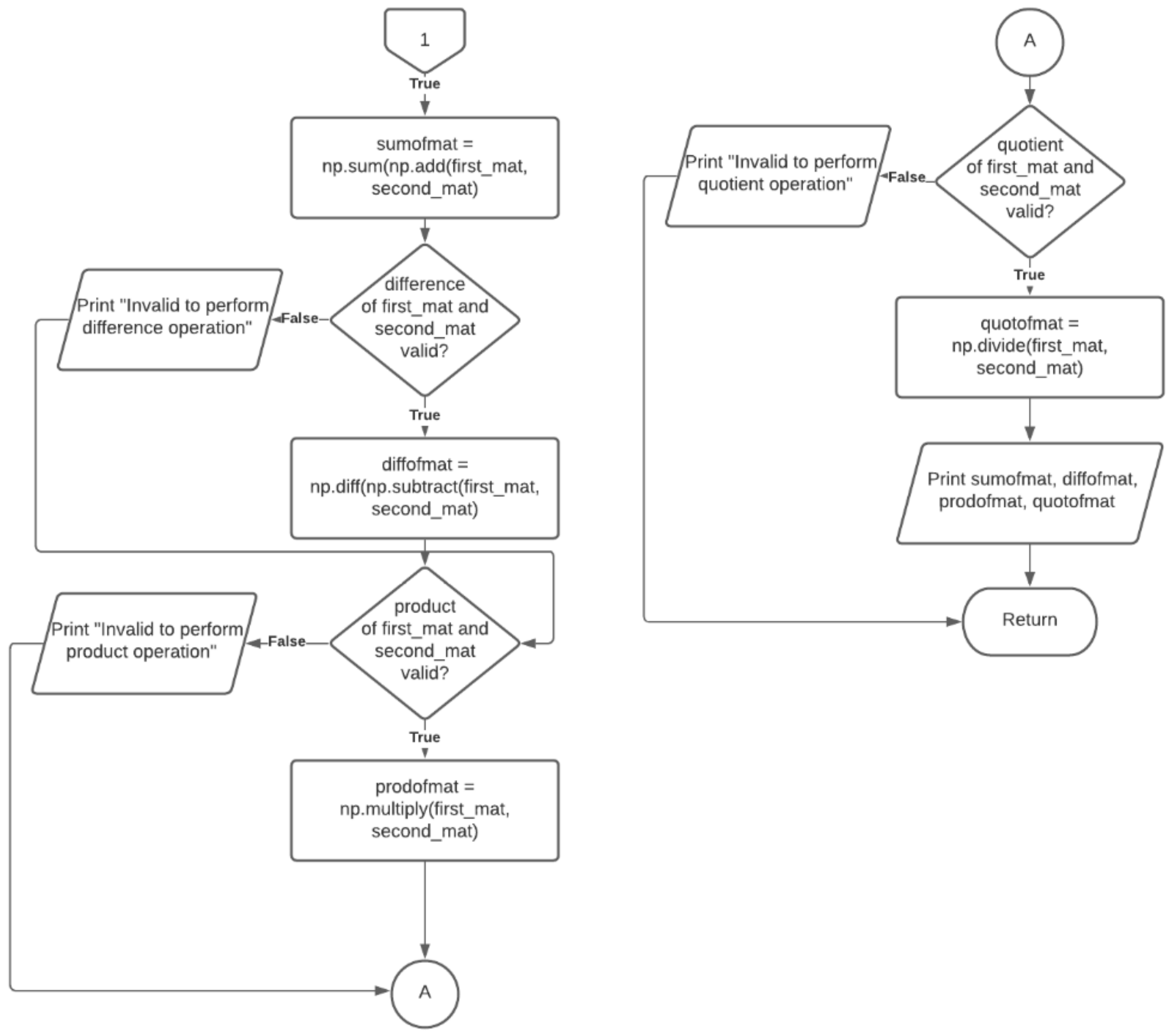

1
True
sumofmat = np.sum(np.add(first_mat, second_mat)
difference of first_mat and second_mat valid?
False
Print "Invalid to perform difference operation"
True
diffofmat = np.diff(np.subtract(first_mat, second_mat)
product of first_mat and second_mat valid?
False
Print "Invalid to perform product operation"
True
prodofmat = np.multiply(first_mat, second_mat)
A
A
quotient of first_mat and second_mat valid?
False
Print "Invalid to perform quotient operation"
True
quotofmat = np.divide(first_mat, second_mat)
Print sumofmat, diffofmat, prodofmat, quotofmat
Return

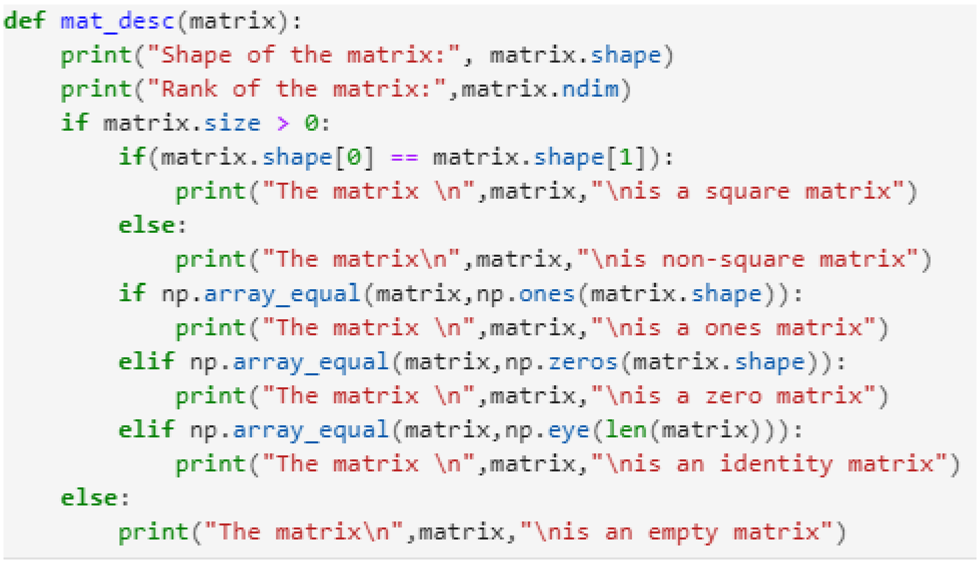
Figure 2.2 Flowchart for task 2

The function first checks if the given parameter is a scalar or a matrix. Using an if-else statement, the function checks if the parameter is an integer or a float or the rank of the parameter is equal to 1. If it is then it will print that the parameter is a scalar, if it is not then it will print the description of the matrix the same as task 1. After determining whether the parameter is a scalar or a matrix, the function will perform basic matrix operations between the first and second parameters. The function used np.sum() and np.add() to compute for the sum between the two parameters [7][8]. Then, the function used np.diff() and np.subtract() to compute for the difference [9][10]. Lastly, the function used np.multiply() and np.divide() to perform element-wise multiplication and element-wise division [11][12]. If the first and second parameters aren't valid to perform matrix operations, instead of an error, it would print an error message using the print function. This is done by using the try and except in Python. First, the try clause will be executed, if the try clause is successful, it would skip the except clause [13].

But if the try clause is unsuccessful then instead of an error, it would output whatever is in the except clause [13].

III. Results

The first task is to create a function named mat_desc() with one parameter that describes the matrix. The codes are seen in figure 3.

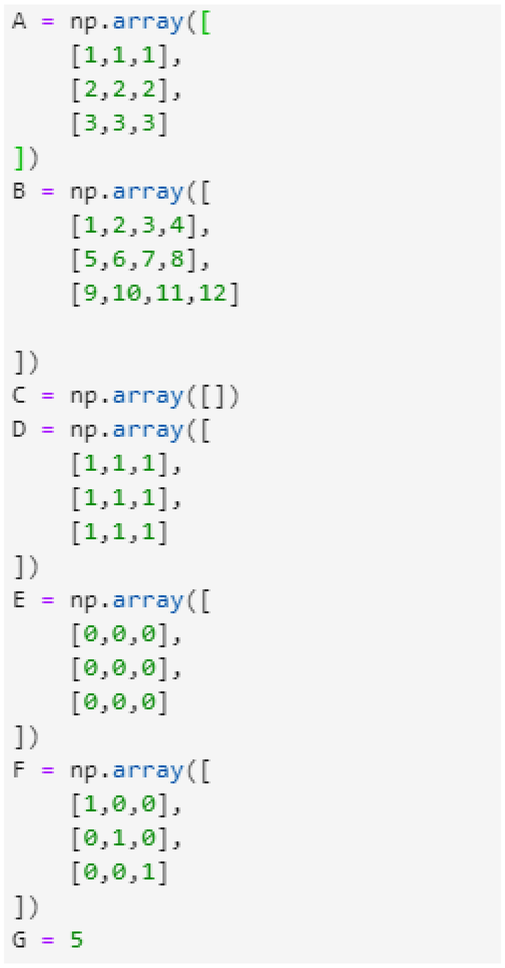


```python
def mat_desc(matrix):
    print("Shape of the matrix:", matrix.shape)
    print("Rank of the matrix:",matrix.ndim)
    if matrix.size > 0:
        if(matrix.shape[0] == matrix.shape[1]):
            print("The matrix \n",matrix,"\nis a square matrix")
        else:
            print("The matrix\n",matrix,"\nis non-square matrix")
        if np.array_equal(matrix,np.ones(matrix.shape)):
            print("The matrix \n",matrix,"\nis a ones matrix")
        elif np.array_equal(matrix,np.zeros(matrix.shape)):
            print("The matrix \n",matrix,"\nis a zero matrix")
        elif np.array_equal(matrix,np.eye(len(matrix))):
            print("The matrix \n",matrix,"\nis an identity matrix")
    else:
        print("The matrix\n",matrix,"\nis an empty matrix")
```

Figure 3 Codes for task 1

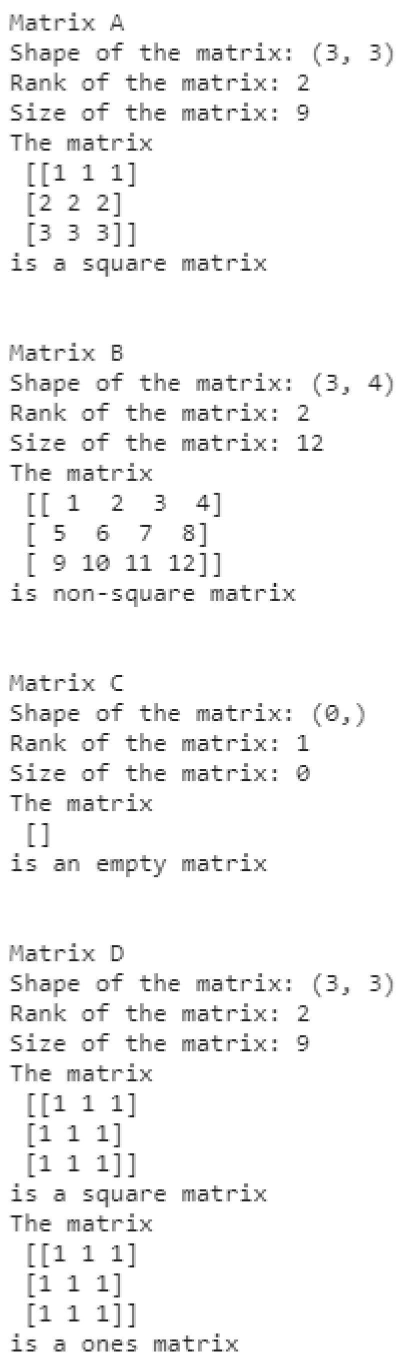
The function describes the matrix's shape, size, and rank. The function also categorizes the matrix. It determines if the parameter is an empty matrix or not. If the parameter is not an empty matrix, it determines whether the matrix is a square or a non-square. Lastly, it determines whether the parameter is an identity, ones, or zeros matrix.

The matrices used for this activity is seen in figure 4. The programmer created 7 variables to test each condition in the created function.



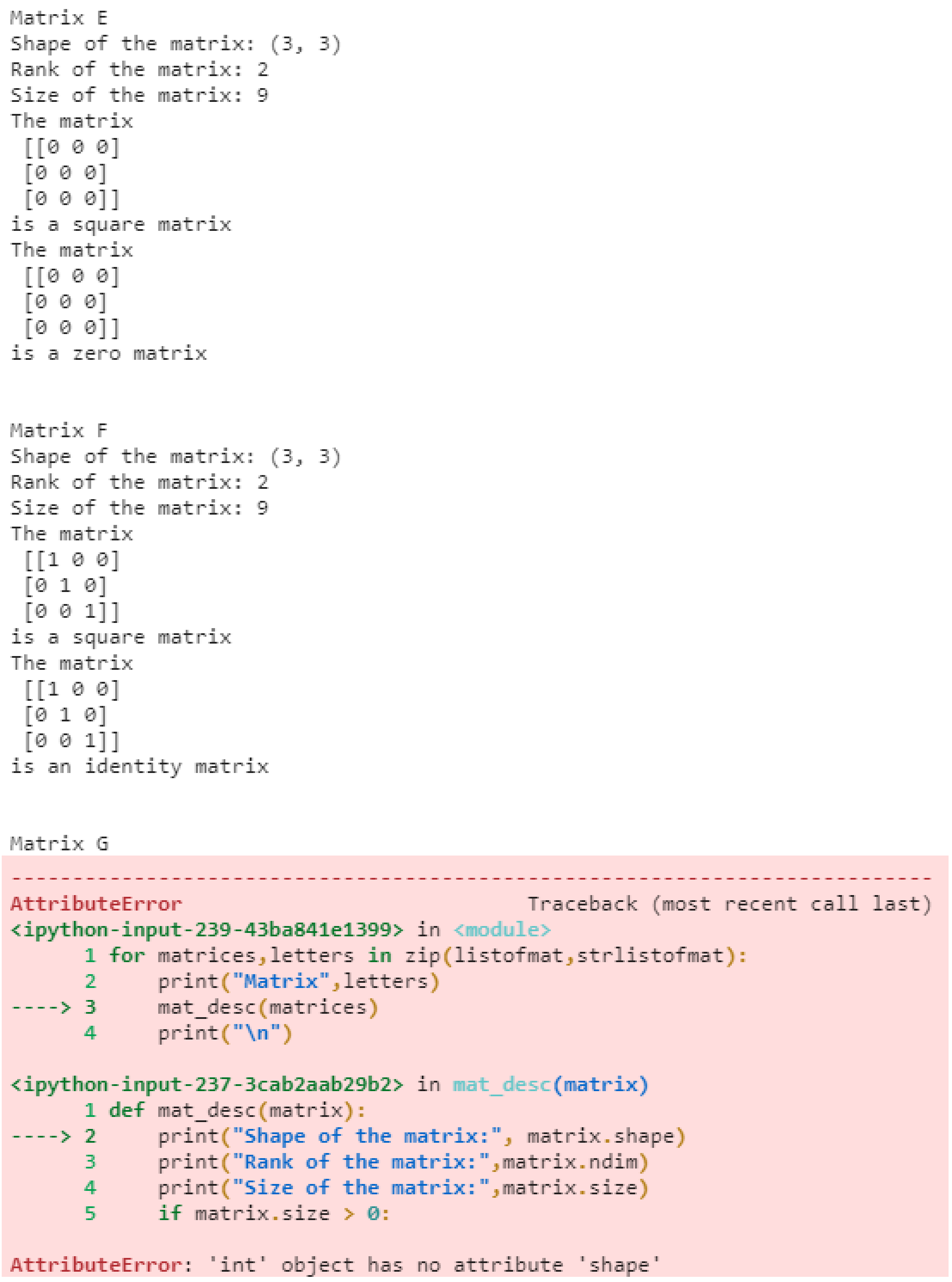
```
A = np.array([
    [1,1,1],
    [2,2,2],
    [3,3,3]
])
B = np.array([
    [1,2,3,4],
    [5,6,7,8],
    [9,10,11,12]

])
C = np.array([])
D = np.array([
    [1,1,1],
    [1,1,1],
    [1,1,1]
])
E = np.array([
    [0,0,0],
    [0,0,0],
    [0,0,0]
])
F = np.array([
    [1,0,0],
    [0,1,0],
    [0,0,1]
])
G = 5
```

Figure 4 Arguments used for the function



```
Matrix A
Shape of the matrix: (3, 3)
Rank of the matrix: 2
Size of the matrix: 9
The matrix
 [[1 1 1]
 [2 2 2]
 [3 3 3]]
is a square matrix


Matrix B
Shape of the matrix: (3, 4)
Rank of the matrix: 2
Size of the matrix: 12
The matrix
 [[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
is non-square matrix


Matrix C
Shape of the matrix: (0,)
Rank of the matrix: 1
Size of the matrix: 0
The matrix
 []
is an empty matrix


Matrix D
Shape of the matrix: (3, 3)
Rank of the matrix: 2
Size of the matrix: 9
The matrix
 [[1 1 1]
 [1 1 1]
 [1 1 1]]
is a square matrix
The matrix
 [[1 1 1]
 [1 1 1]
 [1 1 1]]
is a ones matrix
```

Figure 5.1 Output for task 1

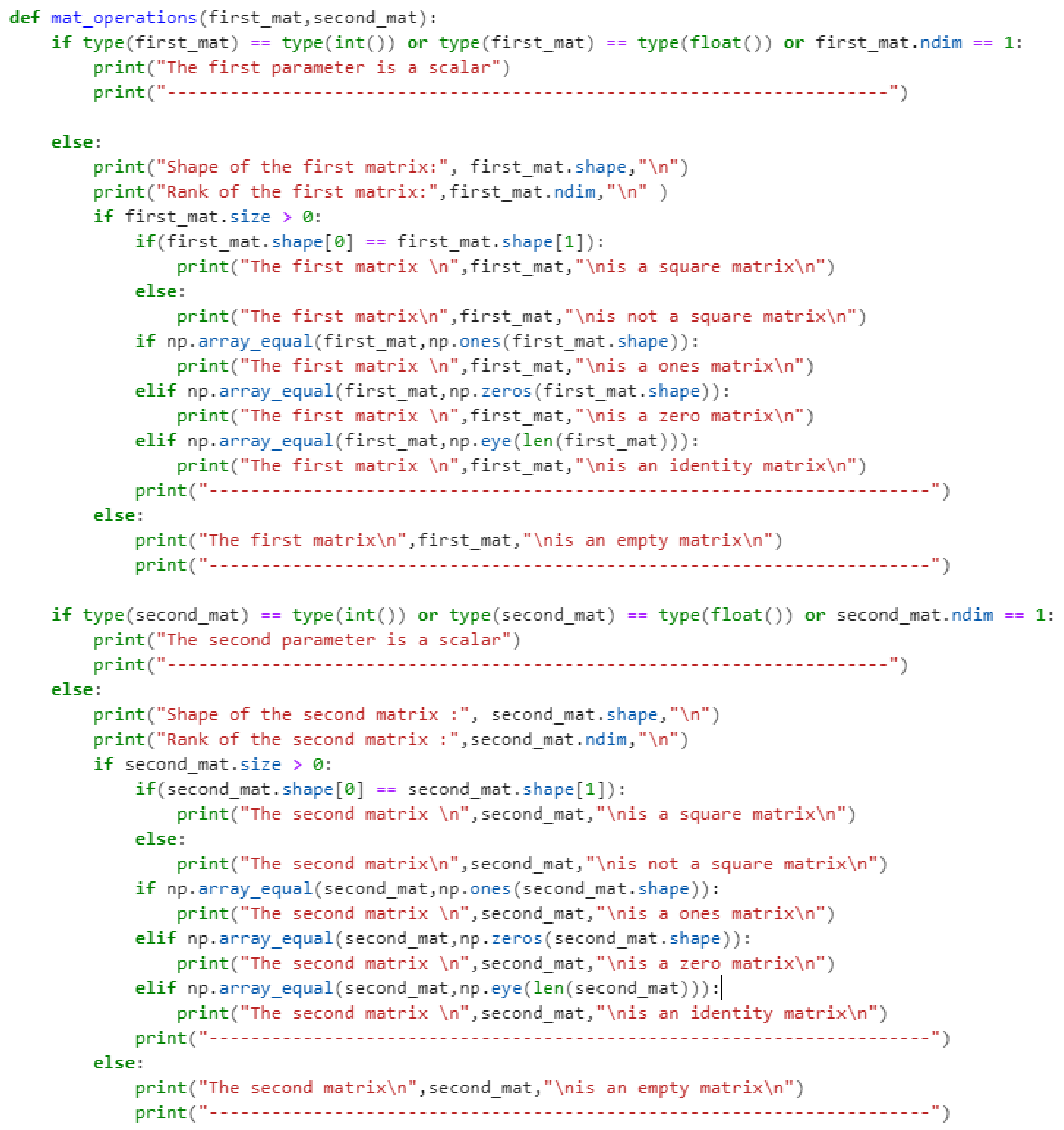


```
Matrix E
Shape of the matrix: (3, 3)
Rank of the matrix: 2
Size of the matrix: 9
The matrix
 [[0 0 0]
 [0 0 0]
 [0 0 0]]
is a square matrix
The matrix
 [[0 0 0]
 [0 0 0]
 [0 0 0]]
is a zero matrix


Matrix F
Shape of the matrix: (3, 3)
Rank of the matrix: 2
Size of the matrix: 9
The matrix
 [[1 0 0]
 [0 1 0]
 [0 0 1]]
is a square matrix
The matrix
 [[1 0 0]
 [0 1 0]
 [0 0 1]]
is an identity matrix


Matrix G
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-239-43ba841e1399> in <module>
      1 for matrices,letters in zip(listofmat,strlistofmat):
      2     print("Matrix",letters)
----> 3     mat_desc(matrices)
      4     print("\n")

<ipython-input-237-3cab2aab29b2> in mat_desc(matrix)
      1 def mat_desc(matrix):
----> 2     print("Shape of the matrix:", matrix.shape)
      3     print("Rank of the matrix:",matrix.ndim)
      4     print("Size of the matrix:",matrix.size)
      5     if matrix.size > 0:

AttributeError: 'int' object has no attribute 'shape'
```

Figure 5.2 Output for task 1

As seen in figures 5.1 and 5.2, the function described each variable accurately. The variable G returned an error due to the value assigned to G being a scalar and not a matrix.

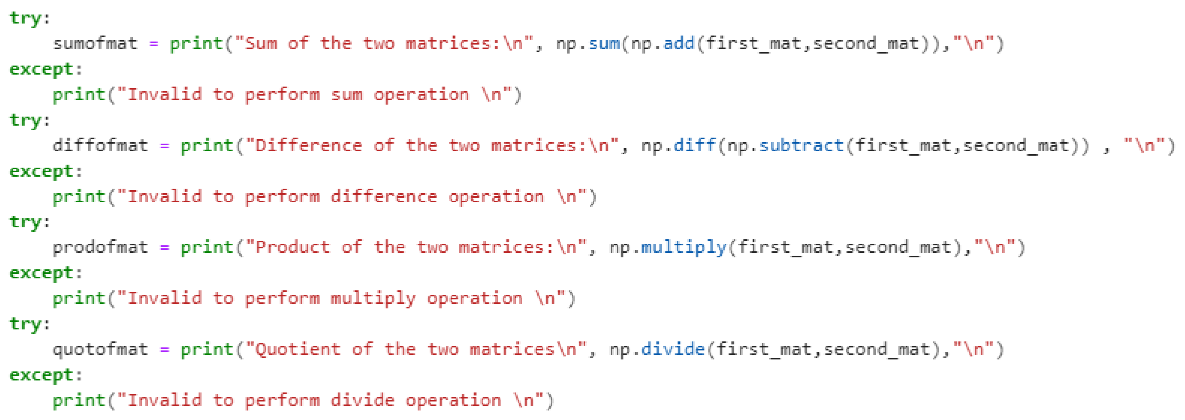
The second task is to create a function named mat_opeartions() with two parameters.



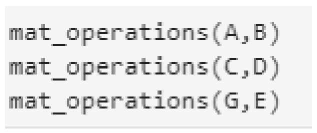
```
def mat_operations(first_mat,second_mat):
    if type(first_mat) == type(int()) or type(first_mat) == type(float()) or first_mat.ndim == 1:
        print("The first parameter is a scalar")
        print("---------------------------------------------------------------------")

    else:
        print("Shape of the first matrix:", first_mat.shape,"\n")
        print("Rank of the first matrix:",first_mat.ndim,"\n" )
        if first_mat.size > 0:
            if(first_mat.shape[0] == first_mat.shape[1]):
                print("The first matrix \n",first_mat,"\nis a square matrix\n")
            else:
                print("The first matrix\n",first_mat,"\nis not a square matrix\n")
            if np.array_equal(first_mat,np.ones(first_mat.shape)):
                print("The first matrix \n",first_mat,"\nis a ones matrix\n")
            elif np.array_equal(first_mat,np.zeros(first_mat.shape)):
                print("The first matrix \n",first_mat,"\nis a zero matrix\n")
            elif np.array_equal(first_mat,np.eye(len(first_mat))):
                print("The first matrix \n",first_mat,"\nis an identity matrix\n")
            print("---------------------------------------------------------------------")
        else:
            print("The first matrix\n",first_mat,"\nis an empty matrix\n")
            print("---------------------------------------------------------------------")

    if type(second_mat) == type(int()) or type(second_mat) == type(float()) or second_mat.ndim == 1:
        print("The second parameter is a scalar")
        print("---------------------------------------------------------------------")
    else:
        print("Shape of the second matrix :", second_mat.shape,"\n")
        print("Rank of the second matrix :",second_mat.ndim,"\n")
        if second_mat.size > 0:
            if(second_mat.shape[0] == second_mat.shape[1]):
                print("The second matrix \n",second_mat,"\nis a square matrix\n")
            else:
                print("The second matrix\n",second_mat,"\nis not a square matrix\n")
            if np.array_equal(second_mat,np.ones(second_mat.shape)):
                print("The second matrix \n",second_mat,"\nis a ones matrix\n")
            elif np.array_equal(second_mat,np.zeros(second_mat.shape)):
                print("The second matrix \n",second_mat,"\nis a zero matrix\n")
            elif np.array_equal(second_mat,np.eye(len(second_mat))):
                print("The second matrix \n",second_mat,"\nis an identity matrix\n")
            print("---------------------------------------------------------------------")
        else:
            print("The second matrix\n",second_mat,"\nis an empty matrix\n")
            print("---------------------------------------------------------------------")
```

Figure 6.1 Codes for task 2

The function determines whether the first or second parameter is a scalar or a matrix. Then it will describe the parameter if it is a matrix. If not then it will print that the parameter is a scalar.



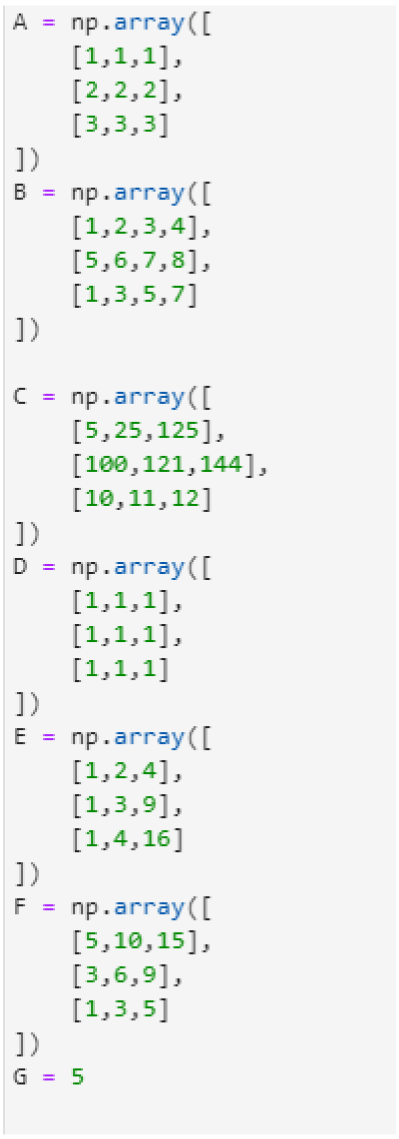
```
try:
    sumofmat = print("Sum of the two matrices:\n", np.sum(np.add(first_mat,second_mat)),"\n")
except:
    print("Invalid to perform sum operation \n")
try:
    diffofmat = print("Difference of the two matrices:\n", np.diff(np.subtract(first_mat,second_mat)) , "\n")
except:
    print("Invalid to perform difference operation \n")
try:
    prodofmat = print("Product of the two matrices:\n", np.multiply(first_mat,second_mat),"\n")
except:
    print("Invalid to perform multiply operation \n")
try:
    quotofmat = print("Quotient of the two matrices\n", np.divide(first_mat,second_mat),"\n")
except:
    print("Invalid to perform divide operation \n")
```



```
mat_operations(A,B)
mat_operations(C,D)
mat_operations(G,E)
```

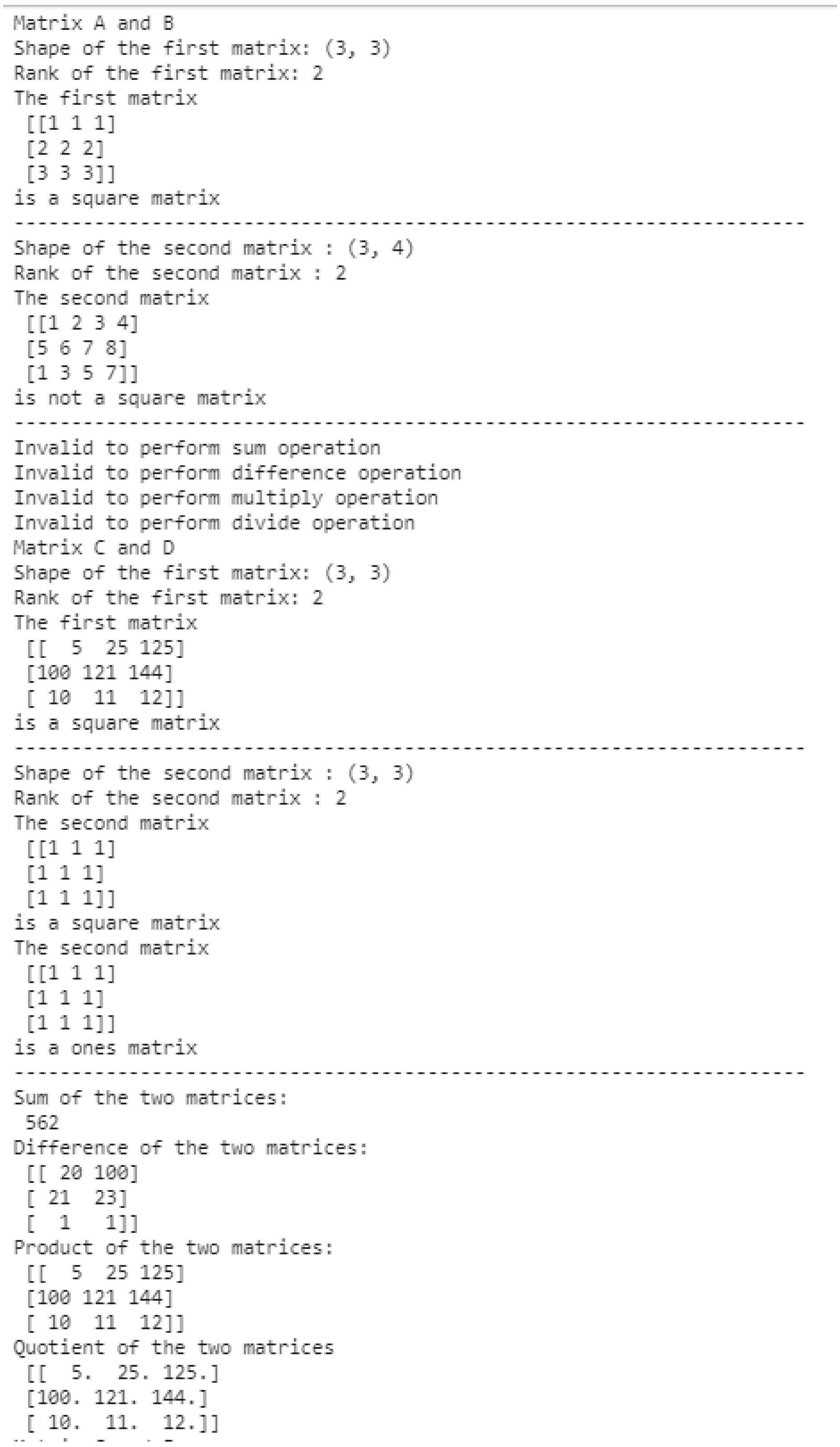
Figure 6.2 Codes for task 2

Next, the function will perform basic matrix operations. Using the try and except statement, if the operation is valid then it will execute and print the answer. If not, instead of returning an error the function will return its own error message “Invalid to perform operation”.



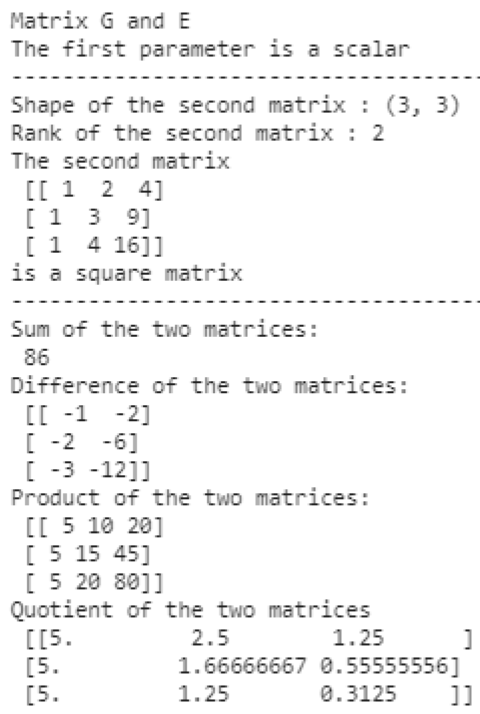
```
A = np.array([
    [1,1,1],
    [2,2,2],
    [3,3,3]
])
B = np.array([
    [1,2,3,4],
    [5,6,7,8],
    [1,3,5,7]
])

C = np.array([
    [5,25,125],
    [100,121,144],
    [10,11,12]
])
D = np.array([
    [1,1,1],
    [1,1,1],
    [1,1,1]
])
E = np.array([
    [1,2,4],
    [1,3,9],
    [1,4,16]
])
F = np.array([
    [5,10,15],
    [3,6,9],
    [1,3,5]
])
G = 5
```

Figure 7 Arguments used for mat_operations() function



```
Matrix A and B
Shape of the first matrix: (3, 3)
Rank of the first matrix: 2
The first matrix
 [[1 1 1]
 [2 2 2]
 [3 3 3]]
is a square matrix
--------------------------------------------------------------------
Shape of the second matrix : (3, 4)
Rank of the second matrix : 2
The second matrix
 [[1 2 3 4]
 [5 6 7 8]
 [1 3 5 7]]
is not a square matrix
--------------------------------------------------------------------
Invalid to perform sum operation
Invalid to perform difference operation
Invalid to perform multiply operation
Invalid to perform divide operation
Matrix C and D
Shape of the first matrix: (3, 3)
Rank of the first matrix: 2
The first matrix
 [[  5  25 125]
 [100 121 144]
 [ 10  11  12]]
is a square matrix
--------------------------------------------------------------------
Shape of the second matrix : (3, 3)
Rank of the second matrix : 2
The second matrix
 [[1 1 1]
 [1 1 1]
 [1 1 1]]
is a square matrix
The second matrix
 [[1 1 1]
 [1 1 1]
 [1 1 1]]
is a ones matrix
--------------------------------------------------------------------
Sum of the two matrices:
 562
Difference of the two matrices:
 [[ 20 100]
 [ 21  23]
 [  1   1]]
Product of the two matrices:
 [[  5  25 125]
 [100 121 144]
 [ 10  11  12]]
Quotient of the two matrices
 [[  5.  25. 125.]
 [100. 121. 144.]
 [ 10.  11.  12.]]
```

Figure 8.1 Output for task 2



```
Matrix G and E
The first parameter is a scalar
--------------------------------------------------------------------
Shape of the second matrix : (3, 3)
Rank of the second matrix : 2
The second matrix
 [[ 1  2  4]
 [ 1  3  9]
 [ 1  4 16]]
is a square matrix
--------------------------------------------------------------------
Sum of the two matrices:
 86
Difference of the two matrices:
 [[ -1  -2]
 [ -2  -6]
 [ -3 -12]]
Product of the two matrices:
 [[ 5 10 20]
 [ 5 15 45]
 [ 5 20 80]]
Quotient of the two matrices
 [[5.         2.5        1.25      ]
 [5.         1.66666667 0.55555556]
 [5.         1.25       0.3125    ]]
```

Figure 8.2 Output for task 2

Figure 8.2 shows the output using the created function mat_operations(). The function categorized the parameter if it is a scalar or a matrix. The function also determined if the matrix is empty or not, square or non-square and identity, ones, or zeros matrix. Lastly, the function performed basic matrix operations using the two parameters and printed the result. If the operation is invalid, it would print its own error message.

IV. Conclusion

The laboratory activity discussed transforming equations into matrices. The activity also discussed the different categories of matrices. Lastly, the activity showed basic matrix operations using Python and NumPy.

Through the activity, the students were able to learn the different functions and techniques used to categorized matrices and do matrix operations using Python and NumPy.

Using the knowledge from this activity and past activities, matrices can be used to solve problems in the field of agriculture. A farmer can calculate the amount of time he needs to completely seed and harvest based on the size of his land and the materials he has. He can also compute the expenses that he will be spending and the profits that he will be able to gain.

References

[1] NumPy, “numpy.ndarray.shape,” 2020. https://numpy.org/doc/stable/reference/generated/numpy.ndarray.shape.html (accessed Mar. 11, 2020).

[2] NumPy, “np.ndarray.size,” 2020. https://numpy.org/doc/stable/reference/generated/numpy.ndarray.size.html (accessed Mar. 11, 2020).

[3] NumPy, “numpy.ndarray.ndim,” 2020. https://numpy.org/doc/stable/reference/generated/numpy.ndarray.ndim.html (accessed Mar. 11, 2020).

[4] NumPy, “numpy.eye,” 2020. .

[5] NumPy, “numpy.eye,” 2020. https://numpy.org/doc/stable/reference/generated/numpy.eye.html (accessed Mar. 11, 2020).

[6] NumPy, “numpy.zeros,” 2020. https://numpy.org/devdocs/reference/generated/numpy.zeros.html (accessed Mar. 11, 2020).

[7] NumPy, “numpy.ones,” 2020. https://numpy.org/doc/stable/reference/generated/numpy.ones.html#numpy-ones (accessed Mar. 11, 2020).

[8] NumPy, “numpy.sum,” 2020. https://numpy.org/doc/stable/reference/generated/numpy.sum.html#numpy-sum.

[9] NumPy, “numpy.add,” 2020. https://numpy.org/doc/stable/reference/generated/numpy.add.html (accessed Mar. 11, 2020).

[10] NumPy, “numpy.diff,” 2020. https://numpy.org/doc/stable/reference/generated/numpy.diff.html (accessed Mar. 11, 2020).

[11] NumPy, “numpy.subtract.” https://numpy.org/doc/stable/reference/generated/numpy.subtract.html (accessed Mar. 11, 2020).

[12] NumPy, “np.multiply,” 2020. https://numpy.org/doc/stable/reference/generated/numpy.multiply.html (accessed Mar. 11, 2020).

[13] NumPy, “np.divide,” 2020. https://numpy.org/doc/stable/reference/generated/numpy.divide.html (accessed Mar. 11, 2020).

[14] Python, “Errors and Exceptions,” 2020. https://docs.python.org/3/tutorial/errors.html (accessed Mar. 11, 2020).

Appendix

Github Repository Link: https://github.com/Loreynszxc/Linear-Algebra-Lab-6