Durham University

International Study Centre

SUMMATIVE ASSIGNMENT FRONT COVER SHEET

Student ID	2595161
CIS Username	qwwk95
Programme	IFY Computer Science
Module	Programming Techniques
Teaching Group	SFSCS_PT
Tutor	Chris Roberts

Assignment Title	F_PT_S_A1:
Assignment Deadline	2021-03-12

It is very important that any work you present as yours must in fact be your own work, and not taken from another place or done by another person. Cheating, collusion (working together with another person on an assessment which is not intended to be collaborative) and copying from unacknowledged sources (plagiarism) are all serious offences and must be avoided.

DEFINITION OF ACADEMIC IMPROPRIETY
Academic impropriety is a term that covers cheating, attempts to cheat, plagiarism, collusion and any other attempts to gain an unfair advantage in assessments. Assessments include all forms of written work, presentations and all forms of examination. Academic impropriety, in any form, is a serious offence and the penalties imposed would reflect this.

DECLARATION
By entering my Student ID below I confirm that this piece of work is a result of my own work except where it forms an assessment based on group project work. In the case of a group project, the work has been prepared in collaboration with other members of the group. Material from the work of others not involved in the project has been acknowledged and quotations and paraphrases suitably indicated. Furthermore, I confirm that I understand the definition of Academic Impropriety that is used by Durham University International Study Centre.

Student ID: 2595161	Date: 2021-03-12

IFY Computer Science

F_PT_S_A1:
Implementing Matrix Operations in Python

2595161

2021-03-12

Contents

1 Introduction

This is a technical document about my Programming Techniques summative assignment 1. The program I created is able to fully meet the matrix operations task specification, as well as some additional functions.

I have added some new functions into the program, including:

- Edit the matrices
- Save the matrices for the next loop
- Calculate matrices from 2x2 to 10x10
- Check which operation system is the user working on and clear screen
- Check are matrices empty
- Handling exceptions and return an error message

2 Task Specification

The minimum specification below must be met:

- be console/text-based
- use nested lists and loops
- include user defined functions
- provide a basic menu system to allow the user to:
  - Enter and store the data for a 3×3 matrix
  - Find and display the sum of two 3×3 matrices
  - Find and display the scalar product of a 3×3 matrix
  - Find and display the product of two 3×3 matrices
  - Exit the program in a controlled manner
- all code must be appropriately commented

3 Software implementation

I have created a menu system as the main program, with different functions. The code could be found in appendix A listing 1

3.1 Pseudo-code Algorithms

A pseudo-code algorithms for each of the matrix operations were created during the planning process, before starting to code the program. As shown below:

```
// first_matrix = First 3x3 matrix
// second_matrix = Second 3x3 matrix
// result = A 3x3 matrix with all 0 in it, to store the result after the
   calculation
// scalar = input(  Enter   the scalar value:    )
// first_colums = 3
// second_rows = 3

Matrix Addition
function matrix_addition()
  for i=0 to len(first_matrix)
    for j=0 to len(first_matrix[0])
      result[i][j] = first_matrix[i][j] + Y[i][j]
    next j
next i
  return result
endfunction

Scalar Multiplication
function scalar_multi()
  for i=0 to len(first_matrix)
    for j=0 to len(first_matrix[0])
      result[i][j] = first_matrix[i][j] * scalar
    next j
next i
  return result
endfunction

Product of Two Matrices
function matrix_multi()
  if first_columns == second_rows then
    for i=0 to len(first_matrix)
      for j=0 to len(second_matrix[0])
        for k=0 to len(second_matrix)
result[i][j] += first_matrix[i][k] * second_matrix [k][j]
        next k
      next j
next i
  else
print(  The   number of columns in the first matrix must equal to the
   number of rows in the second matrix.   )
  endif
  return result
endfunction
```

3.2 Set up

[7-9] Imported necessary modules to the program, as they will be used later on in the program.

[12-13] Set up the lists for the text of numbers in English and the menu options. These lists are useful, as we could get specified element in the list through the list index.

[16-19] Created empty lists to store the value for each matrix. Two extra lists were created as well, because one of them will be storing the result after the calculation and the other one will be used for the matrices to display a '0' matrix, instead of an empty matrix, letting the user visualise the matrix in an easier way.

[22-24] Set up boolean value for variables, these will be used to check whether data entry is completed and is the matrices were checked.

[27-30] Created variable to store messages. These will be used in the main program later on, and some of them will be updated throughout the program. **user_choice** was set up to declare this variable, as it will be used to monitor the loop in the main program.

3.3 Main program - Loop

[304-682] Used while loop for the main program, when **user_choice** is not equal to '5' and this condition is true, the while loop will keep continuing to iterate, until the condition become false. Which is **user_choice** equals to 5.

[307-309] Called the **clear_screen()** function from [63-70] to clear the console output each time it started the loop again. Boolean value for these variables will be also reset each time as well.

[312-313] Printed the messages in those variables, to welcome the user to the program.

[316-317] Used a for loop to print the elements in the list of **menu_options**, in order to let the user to choose what action the user should take next.

[320-321, 641-644] To check is the user entering an integer and handle it through exception if an exception occurs. **clear_screen()** function will be called and **valueError_message** will be printed.

[324, 632] To check does the user entered a value that's in range. If not, the else part will be run.

[327-329] If the user chooses option 1, the program will then recall the **clear_screen()** function to clear the console output and print the message to tell the user have chosen option 1.

[332-336] The user entered the sub-menu, and used a while loop for the following part. Sub-menu options were also printed to let the user choose.

[339-341, 446-448] To check is the user entering an integer and handle it through exception if an exception occurs. **clear_screen()** function will be called and **valueError_message** will be printed.

[344-368] If the user chooses option 1 in the sub-menu, the program will then do the following actions. **clear_screen()** function will be called, as well as **create_matrix()** to create the new matrix. Getting the return value from **create_matrix()**, and put it into different variables. Also, using **copy.deepcopy()** to copy all the elements in the first matrix (nested list), to other matrices. Call **enter_matrix()** to allow the user to enter the value for the first matrix. Also **data_complete** will become true at the end of the part, to leave the loop.

[371-381] If the user chooses option 2 in the sub-menu, the program will check if both matrices were empty. If that's the case, it will print a message to remind the user to create a matrix first. **data_complete** will become true at the end of the part, to leave the loop.

[382-387] If both matrices were not empty, the following part of the program will be run. It will call the **clear_screen()** function to clear the screen. And a while loop was used to repeat this part of the program, when the user selected to continue to edit the matrices.

[390-403, 433-435] Another sub-menu will be printed, and ask and check is the user entering an integer and handle it through exception by printing the **valueError_message** and **clear_screen()** function will be called if an exception occurs. The first matrix will be selected if the user chooses option 1, and the program will check is the first matrix empty or not. If the first matrix is empty. The program will print a reminder message and set **data_complete** to true to leave the loop.

[405-410] If the first matrix is not empty, the program will allow the user to edit the first matrix by calling **edit_matrix()** and ask the user for another edit or not by calling **ask_another_edit()**. A return boolean value will then send back and depends on that value to leave the loop or not.

[413-425] The second matrix will be selected if the user chooses option 2, and the program will check is the second matrix empty or not. If the second matrix is empty. The program will print a reminder message and set **data_complete** to true to leave the loop. If the second matrix is not empty, the program will allow the user to edit the second matrix by calling **edit_matrix()** and ask the user for another edit or not by calling **ask_another_edit()**. A return boolean value will then send back and depends on that value to leave the loop or not.

[428-430] The following code will be run if the user didn't enter an integer that's between 1-2.

[438] Update **data_complete** to true and return to the main menu.

[441-443] The following code will be run if the user didn't enter an integer that's between 1-2.

[451-452] Call **countdown()** to print the countdown animation, and reset **invalid_message** to empty. As the user might

[455] Skip the rest of the main loop, and go back to the start of the while loop. As the program shouldn't ask the user, do they want to continue after editing the matrices.

[458-467] If the user chooses option 2 in the main menu, the program will then recall the **clear_screen()** function to clear the console output and print the message to tell the user have chosen option 2. Then the program will check if the first matrix was empty. If that's the case, it will print a message to remind the user to create a matrix first. **data_complete** will become true at the end of the part, to leave the loop.

[470-498] If the second matrix was not empty. Then it will check are **save_matrix** and **same_size_matrices** true, which means the user had run the program once already and saved the matrices. It will also create a second matrix, if the second matrix is empty, and also allow the user to enter the value for the second matrix. It will also check are both matrices in the same size, print an invalid message, return to the start of the loop, if that's the case. If all conditions meet, then it will start the matrix addition by calling **matrix_addition()**.

[501-520] This part of the program will be run when those conditions on line [473] do not meet. It will use **copy.deepcopy()** to copy the nested list from **empty_matrix** to **second_matrix** and **result_matrix**. Also allow the user to enter the value for the second matrix, then it will calculate the sum of those two matrices by calling **matrix_addition()**.

[523-532] If the user chooses option 3 in the main menu, the program will then recall the **clear_screen()** function to clear the console output and print the message to tell the user have chosen option 3. Then the program will check if the first matrix was empty. If that's the case, it will print a message to remind the user to create a matrix first. **data_complete** will become true at the end of the part, to leave the loop.

[535-544] If the first matrix is not empty, this part of the program will be run. And check is **save_matrix** true, if that's the case, it will print a message and call **time_animation()** to perform the delay effect, and calculate the scalar product of the first matrix by calling **scalar_multi()**.

[547-556] If the user chooses option 4 in the main menu, the program will then recall the **clear_screen()** function to clear the console output and print the message to tell the user have chosen option 4. Then the program will check if the first matrix was empty. If that's the case, it will print a message to remind the user to create a matrix first. **data_complete** will become true at the end of the part, to leave the loop.

[557-579] If the second matrix was not empty. Then it will check are **save_matrix** and **same_size_matrices_rc** true, which means the user had run the program once already and saved the matrices. It will also create a second matrix, if the second matrix is empty. It will also check are the number of columns in the first matrix must equal to the number of rows in the second matrix, it that's the case, if will then allow the user to enter the value for the second matrix. If not, it will print an invalid message, and return to the start of the

loop.

[582-596] If all conditions meet, the program will clear **result_matrix** and atomically create a new **result_matrix**. **clear_screen()** function will also be called, and start the matrix multiplication by calling **matrix_multi()**.

[597-629] This part of the program will be run when those conditions on line [560] do not meet. The program will clear **second_matrix**, and allow the user to create a new second matrix. It will also check are the number of columns in the first matrix must equal to the number of rows in the second matrix, if that's the case, it will then allow the user to enter the value for the second matrix. If not, it will print an invalid message, and return to the start of the loop. If all conditions meet, the program will clear **result_matrix** and atomically create a new **result_matrix**. **clear_screen()** function will also be called, and start the matrix multiplication by calling **matrix_multi()**.

[632-639] If the user entered a value that's doesn't meet any of the condition in the main menu, the program will run the following part. Check did the value that the user enter equals to '5', if that's not the case, it will update **invalid_message** to remind the user to enter a value that's between 1-5.

[647-650] Reset **user_choice** to an empty string. As the user might entered '5' at the main menu, and it will still quit the program, if the user entered any value for **continue_choice**, and if didn't reset the variable. Also ask the user to continue the program or not.

[653-677] Check did the user entered a value that will meet these conditions, if that's the case, it will call **clear_screen()** function and ask the user whether they want to save the matrices or not. Also update the **save_matrices** to true or false depending on the user input, and **clear_matrix()** function will be called to clear the matrices if **save_matrices** is false. Update **invalid_message()** and call **countdown()** function to perform a delay.

[680-682] If all conditions do not meet, this part of the program will be run and **user_choice** will be updated to '5'. Then the main while loop will no longer be true and it will leave the main loop.

[685] The user has quitted the program, and print a goodbye message.

3.4 Functions

In my program, I created many functions to reduce redundancy and improve code efficiency.

3.4.1 Create matrix

[108-151] This is the function to create a new matrix.

[111] Used a while loop, and when the condition is true, the following program will continue. Until the user entered a valid input, then the condition no longer is true.

[114-121, 139-141] To ask and check is the user entering an integer and handle it through exception by printing the **valueError_message** and **clear_screen()** function will be called if an exception occurs.

[124-136] **if-else** was used to check is the input out of range or not, as the program is only able to handle a matrix from 2x2 to 10x10. **clear_screen()** function will be called in whatever situation and print the corresponding message. If the input value is in range, **data_complete** will become true and leave the loop.

[144-145] Create the matrix size depending on the input that the user just entered, and using a for loop to add the value '0' into each rows and columns of the matrix.

[148] **print_matrix()** function is called, and it will print the matrix that the user just created.

[151] Return the values in the list to the caller code. These values are used to check whether the two matrices match.

3.4.2 Enter matrix

[154-184] This is the function to allow the user to enter the value for the matrix.

[157-158] Iterate through rows and columns of the matrix.

[161] To reset **data_complete** to false each time when the loop iterate, to ensure the user entered an integer value.

[164] While loop is used to loop the following part.

[167-176] To ask and check is the user entering an integer and handle it through exception by printing the **valueError_message** and **clear_screen()** function will be called if an exception occurs. It will also change **data_complete** to true to leave the loop.

[180-184] **clear_screen()** function will be called to clear the console output and replace the value into the rows and columns that's the user was entering. **.append()** cannot be used here, because rows and columns of the matrix were created in the **create_matrix()** function, and it's a matrix with the value '0' already. Call **print_matrix()** to print the matrix in a nice order.

3.4.3 Edit matrix

[187-236] This is the function to edit the matrix.

[189-192] **clear_screen()** function will be called and while loop is used to loop the following part.

[195] Call **print_matrix()** to print the matrix in a nice order, to remind the user what valued are in the matrix.

[198-202, 234-236] To ask and check is the user entering a integer and handle it through exception by printing the **valueError_message** and **clear_screen()** function will be called if an exception occurs.

[205-209] Check whether the user input is in range and minus one in order to find the correct index in the list.

[212-226] To ask and check is the user entering an integer and handle it through exception by printing the **valueError_message** and **clear_screen()** function will be called if an exception occurs. Also changed **data_complete** to true to leave the loop.

3.4.4 Input scalar

[239-255] This is the function to allow the user to enter the scalar value.

[242] While loop is used to loop the following part.

[245-255] To ask and check is the user entering an integer and handle it through exception by printing the **valueError_message** and **clear_screen()** function will be called if an exception occurs. **clear_screen()** function will be used and return **scalar** to the caller code in the input is a valid input.

3.4.5 Matrix Addition

[258-268] This is the function to calculate the sum of two matrices.

[261-262] Iterate through rows and columns of the matrix.

[265] Add both values of the matrices, and replace the new value into its corresponding rows and columns in the **result_matrix**.

[268] At the end of the loop, call **print_matrix()** to print the matrix in a nice order.

3.4.6 Scalar Product

[271-284] This is the function to calculate the scalar product of a matrix.

[274] Call **input_scalar()** to let the user enter the scalar value for the calculation.

[277-278] Iterate through rows and columns of the matrix.

[281] Multiply the values of the matrix by the scalar value, and replace the new value into its corresponding rows and columns in the **result_matrix**.

[284] At the end of the loop, call **print_matrix()** to print the matrix in a nice order.

3.4.7 Product of Two Matrices

[287-298] This is the function to calculate the product of two matrices.

[290-292] Iterate through rows, columns and rows of the matrix.

[295] Multiply the value of the first matrix by the value of the second matrix, and add the new value into its corresponding rows and columns in the **result_matrix**.

[298] At the end of the loop, call **print_matrix()** to print the matrix in a nice order.

3.4.8 Others

[33-36] Used a for loop to show a time animation to simulate the process that the program is processing, and mainly for delay purposes.

[39-46] Used a while loop to print a countdown on the same line. Used to tell the user how many seconds left before the next action was taken by the program, and to give the user time to read through the messages printed by this function. This function also acts as a time delay. I searched up on how to create a countdown at [1].

[49-60] To print matrix in a nice order, row by row, instead of just displaying a nested list. This function will get some parameters, and print the matrix using a for loop with a message to tell which matrix is it showing now.

[63-70] To check which operation the user is running this program on and clear the console output for clearer console output. I searched this up on [2].

[73-76] This function is used to clear the data in all of the lists, except **empty_matrix**. As I need to use the empty list to show the user a matrix with the value '0'.

[79-83] After checking the first matrix is empty in the main program, this function will be called. It will print messages to remind the user to create the first matrix and send back the **invalid_message** to the caller code.

[86-93] After checking both matrices do not match in the main program, this function will be called. It will print messages to remind the user that both of the matrices do not match and send back the **invalid_message** to the caller code.

[96-105] Asking the user whether they wanted to edit another matrix, and return a boolean value to the caller code depends on the input and **if-else**.

3.5 Problems

When creating this program, I encountered some problems that bothered me. Some problems that I have got, as shown below:

- I tried to use **.copy()** or **[:]** to copy a nested list to a new list, however, it didn't work. Then I realised that I needed to use **copy.deepcopy()** in order to copy a nested list.
- Since my program requires the user to enter values multiple times, I want to avoid terminating the program if there is an error. I tried to figure out whether there is a way to use **try:** and **except:** as a function to reduce redundancy. However, after I searched online and tried to create the function for it myself, I could not find a solution for it, so I gave up and no longer wasted time to solve this problem.
- When the **countdown()** function is running ([39-46]), it displays the countdown, but the user is still able to enter any value during that time, which will then be used in the next input, and will cause an invalid input. For example, when running the

Table 1: Trace table for the sum of two 3x3 matrices

i	j	first matrix (i)(j)	second matrix (i)(j)	result matrix (i)(j)	output
0	0	0	0	0	
	1	-4	5	1	
	2	-3	6	3	
1	0	-2	7	5	
	1	-1	8	7	
	2	1	1	2	
2	0	2	-2	0	
	1	3	9	12	
	2	4	4	8	$\begin{bmatrix} 0 & 1 & 3 \\ 5 & 7 & 2 \\ 0 & 12 & 8 \end{bmatrix}$

countdown(), I entered the value '4'. Then, the value will also be used in the next input, so for the next input, I entered the value '2', and the program will tell you that this is an invalid input because the program thinks you entered '42'. I tried to find a solution, but still didn't find a solution, so I shouldn't spend more time on it.

4 Verification of results

To test the program, I am going to use two 3x3 matrices (as equation 1 shown) to calculate the scalar product of the first matrix, the sum and the product of those two matrices.

$$\begin{bmatrix} 0 & -4 & -3 \\ -2 & -1 & 1 \\ 2 & 3 & 4 \end{bmatrix}, \begin{bmatrix} 0 & 5 & 6 \\ 7 & 8 & 1 \\ -2 & 9 & 4 \end{bmatrix} \tag{1}$$

4.1 Menu

Once you have started the program, you will be able to see a basic menu including 5 options, as appendix B listing 2 have shown. Which met the task specification, to provide a basic menu system to allow the user to do certain actions.

4.2 Create the first matrix

Appendix B listing 3 shows the console output after the user entered '1' in the main menu. Moreover, appendix B listing 4 shows the console output after the user entered '1' in the sub-menu and entered '3' for rows, and '3' for columns, in order to create the first 3x3 matrix as we mentioned above. Appendix B listing 5 shows the console output after the user entered all value for the first matrix, and it is the same as the first matrix in equation 1.

4.3 Matrix addition

After created the first matrix, the user was returned to the main menu by the program (as appendix B listing 2 shown). The user entered '2' to select 'Matrix Addition', and appendix B listing 6 will be shown in the console output. Appendix B listing 7 shows the console output while the user was entering the value for the second matrix. Then the program will calculate the sum of two 3x3 matrices, as table 1 shown. The final answer calculated by the program was shown in appendix B listing 8.

In order to verify the program calculated the correct answer of those two 3x3 matrices, I will perform manual calculations step by step. As equation 2 shown, I am going to add both 3x3 matrices. I am adding those two 3x3 matrices by adding the corresponding entries together, as equation 3 shown. The final answer of my manual calculation (as equation 4 shown) is the same as the program calculation (as appendix B listing 8 shown). Therefore, the program worked well for matrix addition and I programmed the program correctly.

$$\begin{bmatrix} 0 & -4 & -3 \\ -2 & -1 & 1 \\ 2 & 3 & 4 \end{bmatrix} + \begin{bmatrix} 0 & 5 & 6 \\ 7 & 8 & 1 \\ -2 & 9 & 4 \end{bmatrix} \tag{2}$$

$$= \begin{bmatrix} 0+0 & -4+5 & -3+6 \\ -2+7 & -1+8 & 1+1 \\ 2+(-2) & 3+9 & 4+4 \end{bmatrix} \tag{3}$$

$$= \begin{bmatrix} 0 & 1 & 3 \\ 5 & 7 & 2 \\ 0 & 12 & 8 \end{bmatrix} \tag{4}$$

4.4 Scalar multiplication

After the calculation of the matrix addition, the user has chosen to continue the program and saved the matrices that we just created, as appendix B listing 9 shown. Then the user was returned to the main menu by the program and entered '3' to select 'Scalar Multiplication', as appendix B listing 10 shown. Then the console is output shown as appendix B listing 11. The user entered '6' for the scalar value, then the program will calculate the scalar product of the first matrix, as table 2 shown. The final answer calculated by the program was shown in appendix B listing 12.

To verify the program calculated the correct answer of the scalar product of the first matrix,

Table 2: Trace table for the scalar product of a 3x3 matrix

i	**j**	**scalar**	**first matrix (i)(j)**	**result matrix (i)(j)**	**output**
0	0	6	0	0	
	1	6	-4	-24	
	2	6	-3	-18	
1	0	6	-2	-12	
	1	6	-1	-6	
	2	6	1	6	
2	0	6	2	12	
	1	6	3	18	
	2	6	4	24	$\begin{bmatrix} 0 & \text{-24} & \text{-18} \\ \text{-12} & \text{-6} & 6 \\ 12 & 18 & 24 \end{bmatrix}$

I will perform manual calculations step by step. As equation 5 shown, I am going to times the first matrix by 6. Then I multiply each entry in the matrix by the scalar, as equation 6 shown. The final answer of my manual calculation (as equation 7 shown) is the same as the program calculation (as appendix B listing 12 shown). Therefore, the program worked well for finding the scalar product of the first matrix and I programmed the program correctly.

$$\begin{bmatrix} 0 & \text{-4} & \text{-3} \\ \text{-2} & \text{-1} & 1 \\ 2 & 3 & 4 \end{bmatrix} \times 6 \tag{5}$$

$$= \begin{bmatrix} 0 \times 6 & (-4) \times 6 & (-3) \times 6 \\ (-2) \times 6 & (-1) \times 6 & 1 \times 6 \\ 2 \times 6 & 3 \times 6 & 4 \times 6 \end{bmatrix} \tag{6}$$

$$= \begin{bmatrix} 0 & \text{-24} & \text{-18} \\ \text{-12} & \text{-6} & 6 \\ 12 & 18 & 24 \end{bmatrix} \tag{7}$$

4.5 Product of two matrices

After finding the scalar product of the first matrix, the user chosen to continue the program, but this time, we were not saving the matrices that we created, as appendix B listing 13 shown. Then the user was returned to the main menu by the program and re-created the same first matrix as equation 1 shown. The user entered '4' to select 'Product of Two Matrices', as appendix B listing 10 shown. Then the console output shown as appendix B

Table 3: Trace table for the product of two 3x3 matrices

i	**j**	**k**	**first matrix (i)(k)**	**second matrix (k)(j)**	**result matrix (i)(j)**	**output**
0	0	0	0	0	0	
		1	-4	7	-28	
		2	-3	-2	-22	
	1	0	0	5	0	
		1	-4	8	-32	
		2	-3	9	-59	
	2	0	0	6	0	
		1	-4	1	-4	
		2	-3	4	-16	
1	0	0	-2	0	0	
		1	-1	7	-7	
		2	1	-2	-9	
	1	0	-2	5	-10	
		1	-1	8	-18	
		2	1	9	-9	
	2	0	-2	6	-12	
		1	-1	1	-13	
		2	1	4	-9	
2	0	0	2	0	0	
		1	3	7	21	
		2	4	-2	13	
	1	0	2	5	10	
		1	3	8	34	
		2	4	9	70	
	2	0	2	6	12	
		1	3	1	15	
		2	4	4	31	$\begin{bmatrix} -22 & -59 & -16 \\ -9 & -9 & -9 \\ 13 & 70 & 31 \end{bmatrix}$

listing 14. The user entered '3' for rows and '3' for columns in order to create the second 3x3 matrix. Appendix B listing 15 shown while the user was entering the value for the second matrix. Then the program will calculate the product of the two matrices, as table 3 shown. The final answer calculated by the program was shown in appendix B listing 16.

To verify the program calculated the correct answer of the product of two of the matrices, I will perform manual calculations step by step. As equation 8 shown, I am going to times both 3x3 matrices. Then I multiply the entries in rows of the first matrix by column in the second matrix and add, as equation 9 shown. The final answer of my manual calculation (as equation 10 shown) is the same as the program calculation (as appendix B listing 16 shown). Therefore, the program worked well for finding the product of two 3x3 matrices and I programmed the program correctly.

$$\begin{bmatrix} 0 & \text{-4} & \text{-3} \\ \text{-2} & \text{-1} & 1 \\ 2 & 3 & 4 \end{bmatrix} \times \begin{bmatrix} 0 & 5 & 6 \\ 7 & 8 & 1 \\ \text{-2} & 9 & 4 \end{bmatrix} \tag{8}$$

$$= \begin{bmatrix} 0 \times 0 + (-4) \times 7 + (-3) \times (-2) & 0 \times 5 + (-4) \times 8 + (-3) \times 9 & 0 \times 6 + (-4) \times 1 + (-3) \times 4 \\ (-2) \times 0 + (-1) \times 7 + 1 \times (-2) & (-2) \times 5 + (-1) \times 8 + 1 \times 9 & (-2) \times 6 + (-1) \times 1 + 1 \times 4 \\ 2 \times 0 + 3 \times 7 + 4 \times (-2) & 2 \times 5 + 3 \times 8 + 4 \times 9 & 2 \times 6 + 3 \times 1 + 4 \times 4 \end{bmatrix} \tag{9}$$

$$= \begin{bmatrix} \text{-22} & \text{-59} & \text{-16} \\ \text{-9} & \text{-9} & \text{-9} \\ 13 & 70 & 31 \end{bmatrix} \tag{10}$$

4.6 Exit the program

In order to exit the program in a controlled manner, the user is able to enter '5' to select 'Quit the Program', as appendix B listing 17 shown. A goodbye message will be shown in the console output. User is also able to exit the program after each calculation, as the program will ask the user that do they want to continue at the end of the calculation, as appendix B listing 8, 12, 15 shown.

4.7 Bugs

During my testing, I tried as many combinations as possible, and I discovered there was a bug in the program. Although it was a minor issue, I hope I could fix it in the future. I also believe there will be more bugs that are undiscovered yet.

$$\begin{bmatrix} 5 & 0 & 4 \\ \text{-1} & 6 & 3 \end{bmatrix} \times \begin{bmatrix} 5 & 2 & \text{-2} & 1 \\ \text{-4} & 6 & 3 & 7 \\ 8 & 4 & 9 & \text{-3} \end{bmatrix} = \begin{bmatrix} 57 & 26 & 26 & \text{-7} \\ \text{-5} & 46 & 47 & 32 \end{bmatrix} \tag{11}$$

In order to discover the bug, we are going to use the matrices as 11 shown, and you will need to follow the following step:

1. Create the first matrix, a 2 by 3 matrix (Appendix B listing 18)
2. Select to calculate the product of two matrices (Appendix B listing 2)
3. Create the second matrix, a 3 x 4 matrix (Appendix B listing 14)

4. The program will show the product of two matrices (Appendix B listing 20)
5. Continue the program and save the matrices (Appendix B listing 9)
6. Select matrix addition and invalid message append (Appendix B listing 21)
7. Re-create the first matrix to 3 x 4, so both matrices will be the same size (Appendix B listing 22)
8. Select matrix addition and bugs appear (Appendix B listing 6)

So what we have just done was created two different sizes of matrices, and calculated the product of those two matrices. Next, we continue the program and saved those matrices. Then tried to find the sum of those two matrices, however, both matrices were not in the same size and an invalid message popped up. Then we re-created a new first matrix to a size of 3 x 4, which was the same size as the second matrix that we created in the option of 'product of two matrices'. And tried to calculate the sum of those two matrices. However, the program forced us to re-create the second matrix instead of using the saved second matrix and the matrix that we have just re-created.

5 Conclusion

To conclude, this program has met all the minimum specification as the task specification required. It is able to display a basic menu with 5 options and created several separate functions for each matrix operation.

Testing was conducted throughout the creation, as well as in the verification section. To ensure all calculation that was made by the program were correct.

There are some improvement could be made in this program, as shown below:

– Allow user to skip the **countdown()** animation by pressing any keys on the keyboard, in order to reduce the waiting time for the user.

– Check how many matrices exist, for a better user interface. As appendix B listing 9 shown, the current message is showing '**Your matrix/ matrices has/ have been saved.**'. If it is able to check how many matrices are existing, the correct word would be print.

– Remove any input value entered by the user when the **countdown()** is running.

– To fix the bugs as I mentioned above, by adding a new variable **recreated_matrix** to check did the first matrix recreate.

6 References

[1] P. Pedamkar. (Mar 2020) Python countdown timer. Accessed 11 Mar 2021. [Online]. Available: https://www.educba.com/python-countdown-timer/

[2] P. Elance. (Aug 2019) How to clear screen in python? via tutorialspoint website. Accessed 11 Mar 2021. [Online]. Available: https://www.tutorialspoint.com/how-to-clear-screen-in-python

Appendices

A Python code: full listing

Listing 1: Python Code

```python
#Created by 2595161 from Durham University Interntaional Study Centre
#Last editied on 2021/03/11 17:00 GMT
#Copyright  2021 2595161. All rights reserved.
#Programming Techniques - Summative Assignment 1 v1.0

# import modules
import os
import time
import copy

# set up the lists for the text of numbers in English and the menu options
EngNum = ["first", "second", "third", "fourth", "fifth", "sixth", "seventh", "eighth", "ninth", "tenth"]
menu_options = ["Create or Edit Matrix", "Matrix Addition", "Scalar Multiplication", "Product of Two Matrices", "Quit the Program"]

# create empty lists to store the value for each matrix
first_matrix = []
second_matrix = []
result_matrix = []
empty_matrix = []

# set up boolean value used to check if data entry is complete, and if is the matrices was checked
save_matrices = False
same_size_matrices = True
same_size_matrices_rc = True

# create variables for messages
valueError_message = "Oops! That was a text. Please try again with a valid number... \n"
welcome_message = "Welcome to the Matrix Operation Algorithms. \nPlease type the number below."
invalid_message = ""
user_choice = ""

# to display the time delay animation
def time_animation(t):
    for i in range(t):
        time.sleep(1)
        print(".")

# to countdown, get parameters about how many seconds is the countdown going to run, and the message that will show.
```

```python
def countdown(s, message = "Returning to main menu in"):

    # print countdown on the same line, until s is 0
    while s:
        timer = "{:01d}".format(s)
        print(message, timer, end="\r")
        time.sleep(1)
        s -= 1

# to print matrix in rows, get parameters about which matrix is it and so on
def print_matrix(which_matrix, matrix_name = "result", result_text = ""):

    # only clear_screen() when result_test is not empty
    if result_text != "":
        clear_screen()

    # print text to tell user which matrix is it showing now
    print(f"This is your {matrix_name} matrix. {result_text}")

    # using a for loop to display the matrix in rows instead of in a line
    for r in which_matrix:
        print(r)

# to check which operation system is the user running this program on and clear the screen in the cell prompt
def clear_screen():

    # to check is the user using linux or mac. The os.name for lunux and mac is "posix".
    if os.name == "posix":
        _ = os.system("clear")
    else:
        # for other operation system, e.g. window.
        _ = os.system("cls")

# to clear the list of "first_matrix", "second_matrix" and "result_matrix" of any data
def clear_matrices():
    first_matrix.clear()
    second_matrix.clear()
    result_matrix.clear()

# to display message and return value back to the caller code, after check first matrix is empty in the main program
def check_first_matrix():
    invalid_message = f"\nPlease select '[1] {menu_options[0]}' to create the first matrix first. \n"
    print(f"\nYour first matrix is empty. \nPlease return to the main menu and choose '[1] {menu_options[0]}' to create the first matrix and come back later. \n")
    countdown(5)
    return invalid_message

# to display message and return value back to the caller code, telling that both matrices don't meet the requirements in order to run the matrix operation
def not_same_matrices(matrixEqual_message, options_index):
    clear_screen()
    print(f"Please try again... \n{matrixEqual_message} \n")
    print(f"You can choose '[1] {menu_options[0]}' in the main menu to re-create a new first matrix \nOR \nchoose '[{options_index + 1}] {menu_options[options_index]}' again to create the correct second matrix.")
    invalid_message = f"\nSelect '[1] {menu_options[0]}' to re-create a new first matrix \nOR \nSelect '[{options_index + 1}] {menu_options[options_index]}' again to create the correct second matrix. \n"
    time_animation(5)
    countdown(5)
    return invalid_message

# to ask the user whether they wanted to edit another matrix, and return a boolean value back to the caller code
def ask_another_edit():

    another_edit = input("Do you want to edit again? Y/N: \n")

    if another_edit == "Y" or another_edit == "y" or another_edit == "yes" or another_edit == "Yes" or another_edit == "YES":
        clear_screen()
        return False
    else:
        clear_screen()
        return True

# to ask the user to enter the rows and columns of the matrix that the user is creating and check it is a vaild input
def create_matrix(matrix_index, which_matrix, product_matrices = False, rows = None, columns = None, data_complete = False):

    # using while loop for the following part, until the user entered a vaild input, and condition no longer is true
    while not data_complete:

        # test the input for errors
        try:

            # to check is product_matrices true, if will only be true when operating the product of two matrices, and automatically creating the result matrix
            if not product_matrices:

                # asking the user to enter the number of rows and columns
                rows = int(input(f"Enter the number of rows for the {EngNum[matrix_index]} matrix: \n"))
                columns = int(input(f"Enter the number of columns for the {EngNum[matrix_index]} matrix: \n"))

            # checking is the input out of range or not, as the program is only able to handle a matrix from 2x2 to 10x10
            if (rows < 2 or rows > 10) and (columns < 2 or columns > 10):
                clear_screen()
                print("The number of rows and columns you entered are out of range. \nPlease try again by entering a number between 2-10. \n")
            elif rows < 2 or rows > 10:
                clear_screen()
                print("The number of rows you entered is out of range. \nPlease try again by entering a number between 2-10. \n")
            elif columns < 2 or columns > 10:
                clear_screen()
                print("The number of columns you entered is out of range. \nPlease try again by entering a number between 2-10. \n")
            else:
                clear_screen()
                print("Here is your matrix that you have just created.")
                data_complete = True

        # handle the error if an exception occurs, to prevent the program from being terminating
        except ValueError:
            clear_screen()
            print(valueError_message)

    # create the matrix through the input the user just entered, using a for loop to add the value "0" into each rows and columns
    for i in range(rows):
        which_matrix.append([0] * columns)

    # print the matrix with vaule "0" that the user had just set up
    print_matrix(which_matrix, EngNum[matrix_index])

    # return the value of rows and columns in list form
    return [rows, columns]

```

```python
# to allow user to enter and store the value for the matrix
def enter_matrix(matrix_index, which_matrix):

    # iterate through rows and columns of the matrix
    for i in range(len(which_matrix)):
        for j in range(len(which_matrix[0])):

            # reset the variable each time
            data_complete = False

            # using while loop for the following part, until the user entered a vaild input, and condition no longer is true
            while not data_complete:

                # test the input for errors
                try:
                    matrix_value = int(input(f"Enter the number for the {EngNum[matrix_index]} matrix at {EngNum[i]} row, {EngNum[j]} column: \n"))

                    # leave the while loop
                    data_complete = True

                # handle the error if an exception occurs, to prevent the program from being terminating
                except ValueError:
                    clear_screen()
                    print(valueError_message)

            # clear the screen and change the value of a specific item, refered to the index number
            # cannot use .append(), because rows and columns of the matrix were created in the create_matrix(), and the value is defaulted to "0"
            clear_screen()
            which_matrix[i][j] = matrix_value

            # print the real time update of the matrix
            print_matrix(which_matrix, EngNum[matrix_index])

# to edit the matrix, get parameters about which matrix, rows, columns, index is it and so on
def edit_matrix(matrix_index, which_matrix, which_rows, which_columns, data_complete = False):

    clear_screen()

    # using while loop for the following part, until the user entered a vaild input, and condition no longer is true
    while not data_complete:

        # call print_matrix(), to remind the user which matrix are they trying to edit
        print_matrix(which_matrix, EngNum[matrix_index])

        # test the input for errors
        try:

            # asking the user to enter the number of rows and columns
            edit_rows = int(input("\nEnter the row you want to edit: \n"))
            edit_columns = int(input("\nEnter the column you want to edit: \n"))

            # to check is the user input in range or note
            if edit_rows > 0 and edit_rows <= which_rows and edit_columns > 0 and edit_columns <= which_columns:

                # update variables, as it need to refer to the index number of the matrix (list) later on
                edit_rows = edit_rows - 1
                edit_columns = edit_columns - 1

                # test the input for errors
                try:

                    # asking the user to enter the new value and change the value in the specific rows and columns of the matrix
                    edit_value = int(input("\nEnter the new value: \n"))
                    which_matrix[edit_rows][edit_columns] = edit_value

                    # call clear_screen() and print_matrix() to print the matrix, and update the variable, which will leave the loop
                    clear_screen()
                    print_matrix(which_matrix, EngNum[matrix_index])
                    data_complete = True

                # handle the error if an exception occurs, to prevent the program from being terminating
                except ValueError:
                    clear_screen()
                    print(f"\n{valueError_message}")

            # call clear_screen() and print a reminder message to the user when the the previous conditions were not true
            else:
                clear_screen()
                print(f"Please trying again by entering a number between 1-{which_rows} for row and a number between 1-{which_columns} for column. \n")

        # handle the error if an exception occurs, to prevent the program from being terminating
        except ValueError:
            clear_screen()
            print(valueError_message)

# ask user to enter the scalar value for the matrix
def input_scalar(data_complete = False):

    # using while loop for the following part, until the user entered a vaild input, and condition no longer is true
    while not data_complete:

        # test the input for errors
        try:

            # ask the user to enter the scalar value for the matrix and call clear_screen() and return value back to the caller code
            scalar = int(input(f"Enter the scalar value for your matrix: \n"))
            clear_screen()
            return scalar

        # handle the error if an exception occurs, to prevent the program from being terminating
        except ValueError:
            clear_screen()
            print(valueError_message)

# to calculate the sum of two matrices
def matrix_addition():

    # iterate through rows and columns of the matrix
    for i in range(len(first_matrix)):
        for j in range(len(first_matrix[i])):

            # replace the value in the result_matrix after addition
            result_matrix[i][j] = first_matrix[i][j] + second_matrix[i][j]

    # call print_matrix() to print matrix in rows, and send parameters
    print_matrix(result_matrix, result_text = ("\nThe sum of your two matrices."))

# to find the scalar product of the matrix
```

```
def scalar_multi():

    # store the return value from the function to a variable called "scalar"
    scalar = input_scalar()

    # iterate through rows and columns of the matrix
    for i in range(len(first_matrix)):
        for j in range(len(first_matrix[i])):

            # replace the value in the result_matrix after scalar multplication
            result_matrix[i][j] = first_matrix[i][j] * scalar

    # call print_matrix() and pass the text and the result matrix into it
    print_matrix(result_matrix, result_text = ("\nThe scalar product of the matrix."))

# to find the product of two matrices
def matrix_multi():

    # iterate through rows, columns and rows of the matrix
    for i in range(len(first_matrix)):
        for j in range(len(second_matrix[0])):
            for k in range(len(second_matrix)):

                # add the value in the result_matrix after matrix multplication
                result_matrix[i][j] += first_matrix[i][k] * second_matrix[k][j]

    # call print_matrix() and pass the text and the result matrix into it
    print_matrix(result_matrix, result_text = ("\nThe product of two matrices."))


# MAIN PROGRAM

# the main program will keep looping, until condition is false
while (user_choice != 5):

    # call clear_screen(), and reset variables value to false each time the loop starts
    clear_screen()
    data_complete = False
    data_complete_2 = False

    # print the welcome message
```

```python
    print(welcome_message)
    print(invalid_message)

    # print the menu through the list "menu_options" using for loop
    for i in range(0,5):
        print([i+1], menu_options[i])

    # test the input for errors
    try:
        user_choice = int(input("\nEnter your choice: "))

        # check is the user_choice in range
        if 0 < user_choice <= 4:

            # run the following code if user_choice is 1
            if user_choice == 1:
                clear_screen()
                print("You have chosen Create or Edit Matrix. \n")

                # using while loop for the following part, until the user entered a vaild input, and condition no longer is true
                while not data_complete:

                    # print another sub-menu
                    print("[1] Create the first matrix")
                    print("[2] Edit the matrices")

                    # test the input for errors
                    try:

                        create_choice = int(input("\nEnter your choice: "))

                        # run the following code if create_choice is 1
                        if create_choice == 1:

                            # call clear_screen(), and clear the first matrix
                            # clearing first matrix is needed, as after the loop run once, and when the user decided to re-create a new first matrix
                            clear_screen()
                            first_matrix.clear()
                            print("You have chosen Create the first matrix. \n")

                            # create the first matrix by calling creae_matrix(), and it will return the rows and columns as a list, and
```

```
store into the variable
                        first_matrix_rc = create_matrix(0, first_matrix)

                        # sperate the value of rows and columns into different variables
                        first_rows = first_matrix_rc[0]
                        first_columns = first_matrix_rc[1]

                        # deepcopy the first matrix to empty_matrix and result_matrix
                        empty_matrix = copy.deepcopy(first_matrix)
                        result_matrix = copy.deepcopy(first_matrix)

                        # allow the user to enter the value for the first matrix
                        enter_matrix(0, first_matrix)

                        # update the variable, which will leave the loop
                        print("\nFirst matrix created. \n")
                        data_complete = True

                    # run the following code if create_choice is 2
                    elif create_choice == 2:

                        # check if both matrices are empty
                        if first_matrix == [] and second_matrix == []:

                            # call clear_screen() and pritn message
                            clear_screen()
                            print("Your matrices are empty. \nPlease create one first. \n")

                            # update the variable, which will leave the loop
                            data_complete = True
                        else:

                            clear_screen()

                            # using another while loop for the following part, until the user entered a vaild input, and condition no longer is true
                            while not data_complete_2:

                                # print another sub-menu
                                print("[1] Edit the first matrix")
                                print("[2] Edit the second matrix")

                                # test the input for errors
                                try:
                                    edit_choice = int(input("\nEnter your choice: "))

                                    # run the following code if edit_choice is 1
                                    if edit_choice == 1:

                                        # check is the first matrix empty or not and update the variable, which will leave the sub-loop if that's true
                                        if first_matrix == []:
                                            print("\nYour first matrix is empty. \nPlease create one first. \n")
                                            data_complete_2 = True

                                        else:

                                            # call edit_matrix() and maybe run the loop again depending on the return value from the ask_another_edit() function.
                                            edit_matrix(0, first_matrix, first_rows, first_columns)
                                            print("\nEdited first matrix. \n")
                                            data_complete_2 = ask_another_edit()

                                    # run the following code if edit_choice is 2
                                    elif edit_choice == 2:

                                        # check is the second matrix empty or not and update the variable, which will leave the sub-loop if that's true
                                        if second_matrix == []:
                                            print("\nYour second matrix is empty. \nPlease create one first. \n")
                                            data_complete_2 = True

                                        else:

                                            # call edit_matrix() and maybe run the loop again depending on the return value from the ask_another_edit() function.
                                            edit_matrix(1, second_matrix, second_rows, second_columns)
                                            print("\nEdited second matrix. \n")
                                            data_complete_2 = ask_another_edit()

                                    # run the following code when the the previous conditions were not true
                                    else:
                                        clear_screen()
                                        print("\nPlease trying again by entering a number between 1-2.")

                                # handle the error if an exception occurs, to prevent the program from being terminating
                                except ValueError:
                                    clear_screen()
                                    print("\nOops! That was a text. Please try again with a valid number...")

                        # update the variable, which will leave the main loop
                        data_complete = True

                    # run the following code when the the previous conditions were not true
                    else:
                        clear_screen()
                        print("Please trying again by entering a number between 1-2. \n")

                # handle the error if an exception occurs, to prevent the program from being terminating
                except ValueError:
                    clear_screen()
                    print(valueError_message)

            # call countdown() and to clear the string in "invalid_message", as "invalid_message" might have value in there already
            countdown(5)
            invalid_message = ""

            # skip the rest of the code that's in the loop and go back to the start of the loop
            continue

```

```
        # run the following code if user_choice is 2
        elif user_choice == 2:

            clear_screen()
            print("You have chosen the Matrix Addition.")
            time.sleep(1)

            # check if the first matrix is empty, if true, update the variable from the return value of the function and go back to the start of the loop
            if first_matrix == []:
                invalid_message = check_first_matrix()
                continue

            # run the following code when the the previous conditions were not true
            else:

                # check if both variables are true
                if save_matrices is True and same_size_matrices is True:

                    # check if the second matrix is empty
                    if second_matrix == []:

                        # deepcopy a new second matrix from empty matrix, in order to copy the inner lists as well, and update the variables with the rows and columns
                        second_matrix = copy.deepcopy(empty_matrix)
                        second_rows = len(second_matrix)
                        second_columns = len(second_matrix[0])

                        # let user to enter and store the data for the second matrix
                        enter_matrix(1, second_matrix)

                    # check if both conditions are met, if both true, update variables and go back to the start of the loop
                    if first_columns != second_columns and first_rows != second_rows:
                        invalid_message = not_same_matrices("The number of rows and columns in the first matrix must equal to the number of rows and columns in the second matrix.", 1)
                        same_size_matrices = False
                        continue

                    # print message and call time_animation()
                    print("\nUsing saved matrices to calculate...")
                    time_animation(2)

                    # call matrix_addition() to calculate the sum for the matrices and update variable
                    matrix_addition()
                    same_size_matrices = True

                # run the following code when the the previous conditions were not true
                else:

                    # print messages, call time_animation() and clear the second matrix
                    print("Creating new matrices...")
                    time_animation(2)
                    print("Created new matrices. \n")
                    second_matrix.clear()

                    # deepcopy a new second and result matrix from the empty matrix, and update the variables with the rows and columns
                    second_matrix = copy.deepcopy(empty_matrix)
                    result_matrix = copy.deepcopy(empty_matrix)
                    second_rows = len(second_matrix)
                    second_columns = len(second_matrix[0])

                    # let user to enter and store the data for the second matrix
                    enter_matrix(1, second_matrix)

                    # call matrix_addition() to calculate the sum for the matrices and update variable
                    matrix_addition()
                    same_size_matrices = True

        # run the following code if user_choice is 3
        elif user_choice == 3:

            clear_screen()
            print("You have chosen the Scalar Multiplication. \n")
            time.sleep(1)

            # check if the first matrix is empty, if true, update the variable from the return value of the function and go back to the start of the loop
            if first_matrix == []:
                invalid_message = check_first_matrix()
                continue
            else:
```

```python

                # run the following code if save_matrices is true
                if save_matrices is True:
                    print("Getting the first matrix that you have saved...")
                    time_animation(2)

                    # call scalar_multi() to calculate the scalar product of the first saved matrix
                    scalar_multi()
                else:
                    # call scalar_multi() to calculate the scalar product for the first matrix
                    scalar_multi()

        # run the following code if user_choice is 4
        elif user_choice == 4:

            clear_screen()
            print("You have chosen the Product of Two Matrices.")
            time.sleep(1)

            # check if the first matrix is empty, if true, update the variable from the return value of the function and go back to the start of the loop
            if first_matrix == []:
                invalid_message = check_first_matrix()
                continue
            else:

                # check if both conditions are met
                if save_matrices is True and same_size_matrices_rc is True:

                    # check if the second matrix is empty
                    if second_matrix == []:
                        print("\nLet's create the second matrix... \n")

                        # deepcopy a new second matrix from empty matrix, in order to copy the inner lists as well, and update the variables with the rows and columns
                        second_matrix_rc = create_matrix(1, second_matrix)
                        second_rows = second_matrix_rc[0]
                        second_columns = second_matrix_rc[1]

                        # check if the first_columns and second_rows are the same, if true, allow the user to enter value for the second matrix
                        if first_columns == second_rows:
                            enter_matrix(1, second_matrix)

                    # check if the first_columns and second_rows are not the same, update variables and go back to the start of the loop
                    if first_columns != second_rows:
                        invalid_message = not_same_matrices("The number of columns in the first matrix must equal to the number of rows in the second matrix.", 3)
                        same_size_matrices_rc = False
                        continue

                    # result_matrix is not empty, as when creating the first matrix in option [1] deepcopied, so need to clear it and create a new one
                    result_matrix.clear()

                    # call create_matrix() and send arguments to automatically create the result matrix
                    create_matrix(2, result_matrix, True, first_rows, second_columns)

                    # call clear_screen(), time_animation() and print messages
                    clear_screen()
                    print("Please wait while the program is creating the result list...")
                    time_animation(2)
                    print("\nUsing saved matrices to calculate...")
                    time_animation(2)

                    # call matrix_multi() to calculate the product of two matrices, and update variable
                    matrix_multi()
                    same_size_matrices_rc = True
                else:

                    # clear the second matrix, as user might saved it
                    second_matrix.clear()
                    print("\nLet's create the second matrix... \n")

                    # deepcopy a new second matrix from empty matrix, in order to copy the inner lists as well, and update the variables with the rows and columns
                    second_matrix_rc = create_matrix(1, second_matrix)
                    second_rows = second_matrix_rc[0]
                    second_columns = second_matrix_rc[1]

                    # check if the first_columns and second_rows are not the same, and go back to the start of the loop
                    if first_columns != second_rows:
                        invalid_message = not_same_matrices("The number of columns in the first matrix must equal to the number of rows in the second matrix.", 3)
                        continue

                    # result_matrix is not empty, as when creating the first matrix in option [1] deepcopied, so need to clear it and create a new one
                    result_matrix.clear()

                    # call create_matrix() and send arguments to automatically create the result matrix
                    create_matrix(2, result_matrix, True, first_rows, second_columns)

                    # call clear_screen(), time_animation() and print messages
                    clear_screen()
                    print("Please wait while the program is creating the result list...")
                    time_animation(2)

                    # let user to enter and store the data for the second matrix
                    enter_matrix(1, second_matrix)

                    # call matrix_multi() to calculate the product of two matrices, and update variable
                    matrix_multi()
                    same_size_matrices_rc = True

    # run the following code when the the previous conditions were not true
    else:

        # check if user_choice is not 5 and update the variable
        if user_choice != 5:
            invalid_message = "\nPlease trying again by entering a number between 1-5. \n"

            # go back to the start of the loop, instead of running the following program
            continue

# handle the error if an exception occurs, to prevent the program from being terminating
except ValueError:
    invalid_message = f"\n{valueError_message}"
    continue

# reset variable
user_choice = ""

# to confirm is the user want to continue the program
continue_choice = input("\nDo you want to continue the program? Y/N: \n")

# check if user entered any of these conditions
if continue_choice == "Y" or continue_choice == "y" or continue_choice == "yes" or continue_choice == "Yes" or continue_choice == "YES":
    clear_screen()
    print(f"\nYou can always choose '[1] {menu_options[0]}' in the main menu to re-create a new first matrix or edit your previous matrices. \n")

    # ask the user whether they wanted to save the previous matrix or matrices
    save_choice = input("Do you want to save your previous matrix/matrices for the next calculation? Y/N: \n")

    # check if user entered any of these conditions
    if save_choice == "Y" or save_choice == "y" or save_choice == "yes" or save_choice == "Yes" or save_choice == "YES":

        # print message and update variable to True
        print("\nYour matrix/ matrices has/ have been saved. \n")
        save_matrices = True
    else:

        # print message and update variable to False
        print("\nYour matrix/ matrices has/ have been deleted. \n")
        save_matrices = False

        # call clear_matrices() to clear the list "first_matrix", "second_matrix" and "result_matrix" of any data, ready for next data entry
        clear_matrices()

    # set new value to invalid_message, so it will appear in the next loop, and call countdown()
```

```
        invalid_message = "\nIt's great to see you here again :)\n"
        countdown(5, "Restarting program in")

    # run the following code when the the previous conditions were not true
    else:
        # update the variable, and the condition of the while loop will become false, which will leave the loop
        user_choice = 5

# the user has quitted the program
print("\nThank you for using the Matrix Operation Algorithms. \nHope to see you soon. \nBye Bye")

#### END OF MAIN PROGRAM
```

B Console output: example console output

Listing 2: Basic Menu

```
Welcome to the Matrix Operation Algorithms.
Please type the number below.

[1] Create or Edit Matrix
[2] Matrix Addition
[3] Scalar Multiplication
[4] Product of Two Matrices
[5] Quit the Program

Enter your choice:
```

Listing 3: (1) Create or Edit Matrix Menu

```
You have chosen Create or Edit Matrix.

[1] Create the first matrix
[2] Edit the matrices

Enter your choice:
```

Listing 4: (1) Create or Edit Matrix Menu - Create first matrix

```
You have chosen Create the first matrix.

Enter the number of rows for the first matrix:
3
Enter the number of columns for the first matrix:
3
```

Listing 5: (1) Create or Edit Matrix Menu - Create first matrix

```
This is your first matrix.
[0, -4, -3]
[-2, -1, 1]
[2, 3, 4]

First matrix created.

Returning to main menu in 5
```

Listing 6: (2) Matrix Addition

```
You have chosen the Matrix Addition.
Creating new matrices...
.
.
Created new matrices.
```

```
Enter the number for the second matrix at first row, first column:
```

Listing 7: (2) Matrix Addition - Input Value

```
This is your second matrix.
[0, 5, 6]
[0, 0, 0]
[0, 0, 0]
Enter the number for the second matrix at second row, first column:
```

Listing 8: (2) Matrix Addition - Sum of two matrices

```
This is your result matrix.
The sum of your two matrices.
[0, 1, 3]
[5, 7, 2]
[0, 12, 8]

Do you want to continue the program? Y/N:
```

Listing 9: Continue the program - saving the matrices

```
You can always choose '[1] Create or Edit Matrix' in the main menu to re-
    create a new first matrix or edit your previous matrices.

Do you want to save your previous matrix/matrices for the next calculation
    ? Y/N:
y

Your matrix/ matrices has/ have been saved.

Restarting program in 5
```

Listing 10: Return to main menu

```
Welcome to the Matrix Operation Algorithms.
Please type the number below.

It's great to see you here again :)

[1] Create or Edit Matrix
[2] Matrix Addition
[3] Scalar Multiplication
[4] Product of Two Matrices
[5] Quit the Program

Enter your choice:
```

Listing 11: (3) Scalar Multiplication - Ask for scalar value

```
You have chosen the Scalar Multiplication.

Getting the first matrix that you have saved...
.
.
Enter the scalar value for your matrix:
```

Listing 12: (3) Scalar Multiplication - Scalar product of a matrix

```
This is your result matrix.
The scalar product of the matrix.
[0, -24, -18]
[-12, -6, 6]
[12, 18, 24]

Do you want to continue the program? Y/N:
```

Listing 13: Continue the program - not saving the matrices

```
You can always choose ’[1] Create or Edit Matrix’ in the main menu to re-
   create a new first matrix or edit your previous matrices.

Do you want to save your previous matrix/matrices for the next calculation
   ? Y/N:
n

Your matrix/ matrices has/ have been deleted.

Restarting program in 5
```

Listing 14: (4) Prodcut of Two matrices - Create Second Matrix

```
You have chosen the Product of Two Matrices.

Let’s create the second matrix...

Enter the number of rows for the second matrix:
3
Enter the number of columns for the second matrix:
```

Listing 15: (4) Prodcut of Two matrices - Input Value

```
This is your second matrix.
[0, 5, 6]
[7, 8, 1]
[0, 0, 0]
Enter the number for the second matrix at third row, first column:
```

Listing 16: (4) Prodcut of Two matrices - Product of Two Matrices

```
This is your result matrix.
The product of two matrices.
[-22, -59, -16]
[-9, -9, -9]
[13, 70, 31]

Do you want to continue the program? Y/N:
```

Listing 17: Exit the program

```
Welcome to the Matrix Operation Algorithms.
Please type the number below.

[1] Create or Edit Matrix
[2] Matrix Addition
[3] Scalar Multiplication
[4] Product of Two Matrices
[5] Quit the Program

Enter your choice: 5

Do you want to continue the program? Y/N:
n

Thank you for using the Matrix Operation Algorithms.
Hope to see you soon.
Bye Bye
```

Listing 18: The first 2x3 matrix

```
This is your first matrix.
[5, 0, 4]
[-1, 6, 3]

First matrix created.

Returning to main menu in 5
```

Listing 19: The second 3x4 matrix

```
This is your second matrix.
[5, 2, -2, 1]
[-4, 6, 3, 7]
[8, 4, 9, 0]
Enter the number for the second matrix at third row, fourth column:
```

Listing 20: The product of 2x3 and 3x4 matrices

```
This is your result matrix.
```

```
The product of two matrices.
[57, 26, 26, -7]
[-5, 46, 47, 32]

Do you want to continue the program? Y/N:
```

Listing 21: Invalid message

```
Please try again...
The number of rows and columns in the first matrix must equal to the
    number of rows and columns in the second matrix.

You can choose '[1] Create or Edit Matrix' in the main menu to re-create a
     new first matrix
OR
choose '[2] Matrix Addition' again to create the correct second matrix.
.
.
.
.
.
Returning to main menu in 5
```

Listing 22: Re-create the first matrix

```
This is your first matrix.
[5, 2, -2, 1]
[-4, 6, 3, 7]
[8, 4, 9, -3]

First matrix created.

Returning to main menu in 5
```