Matrix Calculator

Key Features:

- Takes input of two matrices at a time, of same order, as specified by the user, at the beginning
- Gives the user a menu to select an operation to perform on the entered matrices
- Also allows the user to input a pair of matrices as many times as (s)he chooses
- Allows the user to perform addition, subtraction or multiplication on the entered pair of matrices
- The program exits when (s)he want it to.

Limitations:

- Works only for square matrices
- Both the matrices should be of the same order
- The order of matrices can vary from 1x1 up to 5x5
- Can only perform operations on 2 matrices at a time.

Summary of the code:

In the code, everything is sorted into functions. It makes the code easy to understand and debug. First off, everything is placed in an infinite *while loop* that breaks only when the user wants to exit the application, i.e. when (s)he enters **5**. All the choices are placed in a dictionary menu. Decision control structure is used to make a null matrix of the same order as that of the one entered by the user, so that the result is stored in it. This null matrix is named as **result**. There are a total of 5 functions, that have been coded, which are as follows:

- input_mat()
- add_mat()
- sub_mat()
- mult_mat()
- print_mat(lst)

Let's have a closer look at these functions now.

input_mat(): this is the input function. It takes input of one single, square matrix. The number of rows/columns is taken input by the user. To make sure the *rows* is an integer value and that it lies in between the range of 1 to 5 (included), the input of rows is taken in a *while loop* that runs until an acceptable value of *rows* is entered by the user. The **input_mat()** function is also placed inside a *for loop* that iterates 2 times to enter a pair of matrix, which are then printed, in a matrix form, and not in a list. To sort this out, **print_mat(lst)** function is used.

add_mat(): this addition function is based upon the index slicing of the lists. Both the elements of the same index of the two matrices are being accessed by nested *for loops*, and then are being added and assigned to the *respective index of the null matrix that* is already defined. It is called when the user chooses the addition operator from the list of operands displayed.

sub_mat(): this works just like the **add_mat()** function, except that here, the elements are subtracted and then assigned, instead of being added. This function is called when the user chooses the subtraction operator from the displayed list.

mult_mat(): in this function, a double nested *for loop* is used to access the values of the two matrices which are then multiplied and added to the respective index of the result. This logic was taken from https://www.geeksforgeeks.org/python-program-multiply-two-matrices/ .

print_mat(lst): this function is just used to print the list in a matrix form. Nested *for loop* is used to print the elements. This function takes an argument of a list.