Here is the code with comments for beginner:

def snail(array):

    ret = [] # create an empty list to store the result

    if array and array[0]: # check if the array is not empty and has at least one row

        size = len(array) # get the number of rows (or columns) in the array

        for n in range((size + 1) // 2): # loop from 0 to half of the size (rounded up)

            for x in range(n, size - n): # loop from n to size - n - 1

                ret.append(array[n][x]) # add the element at row n and column x to the result

            for y in range(1 + n, size - n): # loop from n + 1 to size - n - 1

                ret.append(array[y][-1 - n]) # add the element at row y and column size - n - 1 to the result

            for x in range(2 + n, size - n + 1): # loop from n + 2 to size - n

                ret.append(array[-1 - n][-x]) # add the element at row size - n - 1 and column -x to the result

            for y in range(2 + n, size - n): # loop from n + 2 to size - n - 1

                ret.append(array[-y][n]) # add the element at row -y and column n to the result

    return ret # return the final result

To explain the code for a beginner, let's go through it line by line:

- The first line defines a function named snail that takes an array as an argument. A function is a block of code that can be reused and called by name.

- The second line creates an empty list named ret. A list is a collection of values that can be changed and accessed by index. We will use this list to store the elements of the array in the desired order.

- The third line checks if the array is not empty and has at least one row. This is done by using the logical operator and, which returns True only if both operands are True. An empty array is considered False in Python, so we need to check both array and array[0]. If this condition is False, we skip the rest of the code and return an empty list.

- The fourth line gets the number of rows (or columns) in the array. This is done by using the len function, which returns the length of a list or any other iterable object. For example, len([1, 2, 3]) returns 3. We assume that the array is square, which means it has the same number of rows and columns.

- The fifth line starts a for loop that runs from 0 to half of the size (rounded up). A for loop is a way to repeat a block of code for each value in a sequence. In this case, we use the xrange function to generate a sequence of numbers. The xrange function takes two or three arguments: start, stop, and step. It returns an object that produces numbers from start (inclusive) to stop (exclusive) with a step (default 1). For example, xrange(0, 5) returns 0, 1, 2, 3, 4. The (size + 1) // 2 expression means to divide size by 2 and round up to the nearest integer. For example, if size is 5, then (size + 1) // 2 is 3. We use this expression because we only need to loop until we reach the middle of the array. The variable n will store each value produced by xrange.

- The sixth line starts another for loop that runs from n to size - n - 1. This loop will go over each element in the top row of each layer of the array. A layer is a square ring of elements around the center of the array. For example, in this array:

[[1 ,2 ,3 ,4],

[5 ,6 ,7 ,8],

[9 ,10,11,12],

[13,14,15,16]]

The first layer is:

[[1 ,2 ,3 ,4],

[8 , ,12],

[16,15,14,13]]

The second layer is:

[[6 ,7 ],

[10,11]]

The variable x will store each value produced by xrange.

- The seventh line adds the element at row n and column x to the result list. This is done by using the append method, which takes another value as an argument and adds it