Baseline runtime is around **1270** **milliseconds**.

We already have row 1 done and checked. Now we need to look at diagonal 1. We do this similar to row 1, with 1 number already being used (top left corner) and we check all the permutations of the remaining 3 in the diagonal. This reduces the previous row 2 check of 4! Down to 3!. Next, following the fancy ordering, I checked col 2, which reduces 3! down to 2! only checking positions 7 and 8. Repeating this for column 3 we need to check 9 and 10. Finally, we check the back diagonal, which only has 1 number left, the 11th position, we can check all the different values for 11, and also doing some pre-checks before going to the very end. – this reduces a few milliseconds < 5 additional milliseconds . Then, finally we need to check (do\_end) col1, col4 as well as row2, row 3. These are the final checks needed, outside cols and inside rows. Just doing this, the fancy ordering and prechecks, it reduced the time down to **~70 milliseconds**. Now, for symmetry, copied functions and pasted them with a c\_ before the function name. The c represents the 1 in the corner. I edited the original set of functions to have the 1 on the top row. After doing this, I had a few errors somewhere causing me to getaround 416 different magic squares, so I removed it so the functionality of this symmetry so that the fancy ordering still worked.