Notes: Project 3, Calcudoku.

HOW TO START?

0. Before anything, GO WATCH THIS VIDEO - https://screencast-o-matic.com/watch/crniVNRPQR
I will not be able to help you or answer your questions until you've listened carefully to that video and carefully considered the Project instructions. Some useful tips and ideas below.

1. Read problem description several times. Define any unclear terms.

2. Write down any invariants or properties of input / output / solution. Describe with precision and care.

3. What restrictions are there on any solution? (Any special framework to use?). Describe with precision and care.

4. What is the nature of the transformation? What type in what type out? Any hints about the algorithms to use?

5. Write down any smaller tasks that you KNOW will need to be handled. Probably functions. Look at I/o and the algorithm to use.

6. Put the tasks together (as if solved) in a framework to solve the problem. This is a very high level description of a solution, algorithm. Probably a main program that uses several functions to do the detailed work.

WHAT are the inputs? Lists of cage descriptions? What gets input to the main program?
What are the outputs? Solutions - what do these look like? Look at the sample run for info on this. Inputs given in sample run. Output of a "solution" also given, with additional stats below it.

Then Read again carefully. Any terms unclear? Define them. "Puzzle" "Backtrack" "NxN grid". "Cage" "row/column" "cell" "brute force". "Check" "valid" "populated cage" "Brute force" seems to involve "trying all solutions" until we find the correct one, no special strategy. That gives part of your overall solution algorithm at a very high level. Take input - try all possible solutions until one is found. NOW, what is a "solution" and how do we know we've found one? Figure this out.

WHAT are the inputs? Lists of cage descriptions? What gets input to the main program?
What are the outputs? Solutions - what do these look like? Notice that I repeatedly emphasize the input types and output types for any function, function call.

Rules for any puzzle: **Invariants.** (5x5 table: array as list of lists)

1. Each row can only have the numbers 1 through N with no duplicates

2. Each column can only have the numbers 1 through N with no duplicates

3. The sum of the numbers in any "cage" should equal the number shown in the upper left portion of the cage.
    * intuit: any cage must consist of contiguous cells (any direction, any "edge") - how to state?
    * any row (column) has only digits 1 - 5 (that's the rub!): have to make that work with all of the cages that are contiguous to each other.

4. The cages are disjoint! - but each row and column must contain 1 - 5 only (each row, collumn sums to 15). (Diagonals must sum to 20? Helpful? An observation, check for accuracy.)

5. Format for each puzzle:

number_of_cages
cage_sum number_of_cells (list of cells)
……..

So one of the problems to solve is how to take that input (input returns strings, there will be some blanks and returns?) - and store it so that you can use the information to solve the puzzle. This is a good bite sized task you can get to right away.

Known:

Cells are numbered 0 - 25 (25 cells, it is a 5x5 puzzle) for purposes of solving the puzzle. (I think)

Data structure? List of lists. Row col is what we want to name. Puzzle[row][column]. - how does this thing work?

Initialize the puzzle list of lists to all zeroes
Cells are numbered from 0 to 24 (25 cells, 5x5). - what pattern used to number? 0 for upper left, increase left to right, then from top to bottom. Draw this. (Look at example)

0 1 2 3 4
5 6 7 8 ….
…
24

Store puzzle as list of lists

Store cages as list of lists
Each cage is a list

FUNCTIONS REQUIRED: Describe the I/o for each and the algorithm to achieve its task.

get_cages() : List of cages # returns the list of cages, each cage is a list. The input function.

check_rows_valid(puzzle): boolean. # check that rows are "valid". - that each row contains exactly one of each 1 - 5 ? (Would be nice to do as small function just loop on one row? A function named **check_row_valid(List**): Boolean might be called over and over. May be a lot like check_columns_valid.

check_columns_valid(puzzle): boolean

check_cages_valid(puzzle, cages): boolean

check_valid(puzzle, cages): boolean. From problem description: ' your puzzle is invalid if 1) duplicates exist in any row or column, 2) the sum of values in a fully populated cage does not equal the required sum 3).the sum of values in a partially populated cage exceeds or equals the required sum"

(Describe each task for these functions - hand written algorithm notes. Top down, divide and conquer. Helper functions may be nicely used by other helper functions. Choose function "size" to do one simple task when possible.)

OVERALL algorithm, from "Solving a Puzzle" in the description. It is "brute force"

1. Place a 1 in the current cell (starting with the upper left cell). (Why?)
2. "Check" if the number is valid (???) continue to the next cell to the right (advancing to the next row when necessary.)
3. If the number is not valid (??). increment the number and "check" for validity again
4. If the number is the maximum possible and is still invalid (what does this look like?) , then "backtrack" by setting the current cell to 0 and moving back to the previous cell.

Now, puzzle is invalid if
a) dups exist in any row or column
b) sum of values in a fully populated cage does not equal the required sum
c) the sum of values in a partially populated cage exceeds or equals the required sum (why?)