Q#01

1. The code creates an empty 9x9 grid using a nested list comprehension, where each element is initialized to 0.
2. Function **valid_num** takes in three arguments: **x** and **y**, the row and column indices of the cell to check, and **num**, the number to check and checks if the number **num** is valid to be placed in the cell **(x,y)** by checking if it appears in the same row, column, or 3x3 sub-grid as the cell. If **num** is valid, the function returns **True**; otherwise, it returns **False**.
3. **solve_puzzle** function solves the Sudoku puzzle using a recursive backtracking algorithm. The function iterates through each cell in the grid, and if the cell is empty (contains 0), it tries to place a number from 1 to 9 in the cell. If the number is valid according to the **valid_num** function, it places the number in the cell and recursively calls the **solve_puzzle** function to solve the next cell. If the **solve_puzzle** function returns **True**, it means the puzzle has been solved, so the function returns **True** as well. If the **solve_puzzle** function returns **False**, it means that the number placed in the cell was not valid, so the function backtracks and tries the next number. If all numbers have been tried and none of them are valid, the function returns **False**.
4. The while loop generates random numbers to fill in the grid until a solvable puzzle is generated. The loop generates a 3x3 sub-grid at a time and fills the diagonal cells with 3 random numbers between 1 and 9 using the **random.sample** function. If the resulting puzzle is solvable (i.e., **solve_puzzle** returns **True**), the loop is broken and the generated puzzle is printed.
5. The code prints the generated Sudoku puzzle.
6. The code calls the **solve_puzzle** function again to solve the generated puzzle, and prints the solution.

Q#02

1. An empty 3x3 grid is created using a nested list comprehension, where each element is initialized to 0.
2. The code generates a list of numbers from 1 to 9 using the **random.sample** function, and fills in the grid with the numbers in random order using nested for loops.
3. The code prints the randomly generated puzzle.
4. The code enters a while loop that checks if the puzzle is a valid magic square. It does this by calculating the sum of each row, column, and diagonal using list comprehension and the **zip** function. If all of the sums are equal to the magic sum of 15, the loop is broken and the puzzle is valid. If not, the code generates a new puzzle using the same process as step 4.
5. The code prints the solution (i.e., the valid magic square puzzle).