Sure, let's break this down step-by-step:

1. \*\*Define the Sudoku Solver using Backtracking:\*\*

- Create a function to check if a number can be placed in a given cell.

- Create a function to solve the Sudoku using backtracking.

2. \*\*Generate Sudoku Puzzle:\*\*

- Create a function to generate a complete Sudoku puzzle.

- Create a function to remove numbers from the complete puzzle to create a playable puzzle.

3. \*\*Add Difficulty Levels:\*\*

- Define different difficulty levels by varying the number of cells removed.

Here is the implementation:

```python

import random

def is\_valid(board, row, col, num):

for i in range(9):

if board[row][i] == num or board[i][col] == num:

return False

start\_row, start\_col = 3 \* (row // 3), 3 \* (col // 3)

for i in range(3):

for j in range(3):

if board[start\_row + i][start\_col + j] == num:

return False

return True

def solve\_sudoku(board):

empty = find\_empty\_location(board)

if not empty:

return True

row, col = empty

for num in range(1, 10):

if is\_valid(board, row, col, num):

board[row][col] = num

if solve\_sudoku(board):

return True

board[row][col] = 0

return False

def find\_empty\_location(board):

for i in range(9):

for j in range(9):

if board[i][j] == 0:

return (i, j)

return None

def generate\_complete\_sudoku():

board = [[0] \* 9 for \_ in range(9)]

solve\_sudoku(board)

return board

def remove\_numbers\_from\_board(board, num\_holes):

holes = set()

while len(holes) < num\_holes:

row, col = random.randint(0, 8), random.randint(0, 8)

if (row, col) not in holes:

holes.add((row, col))

board[row][col] = 0

return board

def generate\_sudoku(difficulty='easy'):

complete\_board = generate\_complete\_sudoku()

if difficulty == 'easy':

num\_holes = 30

elif difficulty == 'medium':

num\_holes = 40

elif difficulty == 'hard':

num\_holes = 50

else:

raise ValueError("Invalid difficulty level")

return remove\_numbers\_from\_board(complete\_board, num\_holes)

# Example usage:

difficulty = 'medium'

sudoku\_board = generate\_sudoku(difficulty)

for row in sudoku\_board:

print(row)

```

### Explanation:

1. \*\*is\_valid:\*\* Checks if a number can be placed in a specific cell.

2. \*\*solve\_sudoku:\*\* Solves the Sudoku puzzle using backtracking.

3. \*\*find\_empty\_location:\*\* Finds an empty cell in the board.

4. \*\*generate\_complete\_sudoku:\*\* Generates a complete Sudoku puzzle.

5. \*\*remove\_numbers\_from\_board:\*\* Removes numbers from the complete puzzle to create a playable puzzle.

6. \*\*generate\_sudoku:\*\* Generates a Sudoku puzzle based on the difficulty level.

You can adjust the number of holes for different difficulty levels as needed.