Subject: Computer Programming: Python

Faculty: Premanand S / SENSE / VIT-CC

FAT Question: 4

Kithan, a mathematical genius, and Krishna, a Pythonista, are trying to create an interesting number pattern called the Number Spiral. The goal of this pattern is to fill an n×n matrix, where n is an odd number, with numbers in a spiral pattern, starting from the center and moving outward in a specific direction.

Requirements:

Input: The user must input an odd integer n, where n≥3. This will represent the size of the square matrix (e.g., for n=5, it will create a 5x5 matrix).

Spiral Filling: The numbers must be filled in the matrix starting from the center and spiraling outward in the following order:

Start at the center and place the number 1.

Then, move right, up, left, and down, and continue spiraling in a clockwise direction.

Pattern Generation:

If n=5, the final output should look like this:

21 22 23 24 25

20 7 8 9 10

19 6 1 2 11

18 5 4 3 12

17 16 15 14 13

Validation: The user must input an odd number only. If the input is invalid, prompt the user to input a valid odd number greater than or equal to 3.

Example:

Enter an odd integer (n >= 3): 3

7 8 9

6 1 2

5 4 3

Solution:

Input:

The user is prompted to input an odd integer `n` (greater than or equal to 3), which will represent the size of the square matrix (n×n) that needs to be filled in a spiral pattern.

Enter an odd integer (n >= 3): 5

Output:

The output is a square matrix of size `n×n`, where the numbers are filled in a spiral pattern starting from the center and spiraling outward in a clockwise direction.

For `n = 5`, the output will look like this:

1 2 3 4 5
16 17 18 19 6
15 24 25 20 7
14 23 22 21 8
13 12 11 10 9

Processing:

1. **Input Validation**:

- The user enters a valid odd integer `n` where `n >= 3`. If the input is invalid, the program prompts the user to input a valid integer.

2. **Matrix Initialization**:

- An `n×n` matrix is initialized, with all elements set to 0.

3. **Spiral Filling**:

- The filling starts from the center of the matrix (calculated as `n//2` for both row and column).
- The number `1` is placed at the center.
- The matrix is filled in a spiral pattern by moving in the directions of right, down, left, and up, in a clockwise order, incrementing the step length after every two direction changes.
- Each time a direction is changed, the number to be placed is incremented by 1.

4. **Output**:

- After filling the matrix, the program prints the matrix row by row, aligning the numbers to make the output readable.

Algorithm:

1. **Input Validation**:

- Input a number `n` from the user.
- Ensure `n` is an odd integer greater than or equal to 3. If not, prompt the user to enter a valid input.

2. **Matrix Initialization**:

- Create an `n×n` matrix initialized with zero values.

3. **Spiral Pattern Filling**:

- Start from the center of the matrix `(n//2, n//2)`.
- Assign the number 1 to this position.
- Use a set of movement directions: right `(0, 1)`, down `(1, 0)`, left `(0, -1)`, up `(-1, 0)`.
- Begin filling the matrix with numbers, starting with 2, and continue until `n*n` is filled.
- After each direction is filled, move to the next direction in the sequence.
- Increase the step length after every two direction changes to ensure the spiral grows outward.

4. **Output the Matrix**:

- Print the matrix, ensuring each number is right-aligned for proper formatting.

Solution Alternative:

An alternative solution could involve using a recursive approach to fill the matrix instead of iterating over each position with a fixed step length. The recursive approach could use a helper function that keeps track of the current position, the current direction, and the remaining numbers to be placed. This method would require more sophisticated backtracking, but would provide the same result.

For example:

- Define a recursive function that places a number in the matrix and then calls itself to place the next number in the correct direction.
- If a conflict arises (i.e., an already filled position), backtrack and try a different direction.

However, the current approach using iteration and direction changes in a fixed sequence is simpler and more efficient for this task.

Code:

```
def generate_spiral(n):
    # Create an empty n x n matrix filled with zeros
    matrix = [[0 for _ in range(n)] for _ in range(n)]
```

```
    # Define the movement directions (right, down, left, up)
    directions = [(0, 1), (1, 0), (0, -1), (-1, 0)]  # (row change, col change)

    # Start from the center of the matrix
    row, col = n // 2, n // 2

    # Starting direction is 'right' (index 0 in directions)
    dir_idx = 0

    # Fill the matrix with numbers from 1 to n*n
    matrix[row][col] = 1  # Start with 1 in the center
    num = 2  # Start filling from the next number

    # Initialize the current step length and the direction counter
    step_length = 1  # Length of the first spiral step
    while num <= n * n:
        for _ in range(2):  # Each step length is applied twice: first for right+down, then for
left+up
            for _ in range(step_length):
                row += directions[dir_idx][0]
                col += directions[dir_idx][1]
                if 0 <= row < n and 0 <= col < n:
                    matrix[row][col] = num
                    num += 1
            # Change direction (right -> down -> left -> up -> right -> ...)
            dir_idx = (dir_idx + 1) % 4

        # After completing two directions, increase the step length
        step_length += 1

    return matrix

def print_matrix(matrix):
    for row in matrix:
        print(" ".join(str(num).rjust(2) for num in row))

def main():
    # Input validation: ensure the user enters an odd integer >= 3
    while True:
        try:
            n = int(input("Enter an odd integer (n >= 3): "))
            if n >= 3 and n % 2 != 0:
                break
            else:
                print("Please enter a valid odd integer greater than or equal to 3.")
        except ValueError:
            print("Invalid input. Please enter a valid integer.")
```

```
    # Generate the spiral matrix
    spiral_matrix = generate_spiral(n)

    # Output the spiral matrix
    print_matrix(spiral_matrix)

# Run the main function
if __name__ == "__main__":
    main()
```

Output:

```
Enter an odd integer (n >= 3): 5
21 22 23 24 25
20  7  8  9 10
19  6  1  2 11
18  5  4  3 12
17 16 15 14 13
```