Task 1: Generate a Random Maze and Display It

Explanation:

We will use Python's random library to create a maze layout consisting of walls (#) and paths (.). The maze will be displayed in the console.

Code:

```
import random

def generate_maze(width, height):
    maze = []
    for _ in range(height):
        row = []
        for _ in range(width):
            cell = "#" if random.random() < 0.3 else "."
            row.append(cell)
        maze.append(row)
    return maze

def display_maze(maze):
    for row in maze:
        print(" ".join(row))

# Maze size
maze_width = 10
maze_height = 6
```

maze = generate_maze(maze_width, maze_height)

display_maze(maze)

How it works:

1. We loop over the grid size.

2. Each cell has a 30% chance of becoming a wall (#).

3. The rest are paths (.).

4. The maze is printed in a neat grid.

Task 2: Tic-Tac-Toe (Player vs Player)

Explanation:

Two players take turns marking X or O in a 3×3 board. The game ends when one wins or the board is full.

Code:

```
def print_board(board):
    for row in board:
        print(" | ".join(row))
        print("-" * 5)

def check_winner(board, player):
    for row in board:
        if all(cell == player for cell in row):
            return True
    for col in range(3):
        if all(board[row][col] == player for row in range(3)):
            return True
    if all(board[i][i] == player for i in range(3)) or \
       all(board[i][2-i] == player for i in range(3)):
        return True
    return False

board = [[" " for _ in range(3)] for _ in range(3)]
current_player = "X"

for turn in range(9):
    print_board(board)
    row = int(input(f"Player {current_player}, enter row (0-2): "))
```

```
    col = int(input(f"Player {current_player}, enter col (0-2): "))

    if board[row][col] == " ":
        board[row][col] = current_player
        if check_winner(board, current_player):
            print_board(board)
            print(f"Player {current_player} wins!")
            break
        current_player = "O" if current_player == "X" else "X"
    else:
        print("Cell already taken! Try again.")
else:
    print_board(board)
    print("It's a draw!")
```

How it works:

print_board() shows the board after every move.

check_winner() checks rows, columns, and diagonals.

Players alternate until there’s a winner or draw.

Task 3: Towers of Hanoi using Recursion

Explanation:

Move all disks from the source peg to the destination peg, following the rules:

1. Only one disk can be moved at a time.

2. A disk can only be placed on top of a larger disk.

Code:

```
def hanoi(n, source, target, auxiliary):
    if n == 1:
        print(f"Move disk 1 from {source} to {target}")
    else:
        hanoi(n-1, source, auxiliary, target)
        print(f"Move disk {n} from {source} to {target}")
        hanoi(n-1, auxiliary, target, source)

# Example: 3 disks
hanoi(3, "A", "C", "B")
```

How it works:

Move n-1 disks to the auxiliary peg.

Move the largest disk to the target peg.

Move the smaller stack from auxiliary to target.

Task 4: Python Program to Calculate Factorial

Explanation:

We'll calculate factorial using recursion, where:

n! = n × (n-1)! and 0! = 1.

Code:

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)

num = int(input("Enter a number: "))
print(f"The factorial of {num} is {factorial(num)}")
```

How it works:

Base case: factorial(0) = 1.

Recursive case: multiply n by factorial of n-1.

Task 5: Check Prime Number

Explanation:

A prime number has only two factors: 1 and itself.

Code:

```
def is_prime(n):
    if n <= 1:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True

num = int(input("Enter a number: "))
```

```
if is_prime(num):
    print(f"{num} is a prime number.")
else:
    print(f"{num} is not a prime number.")
```

How it works:

We check divisibility from 2 up to $\sqrt{n}$.

If divisible, it’s not prime.