**Top 10 Coding Questions on Functions & Recursion**

**📌 1. Function Basics: Find the Maximum of Two Numbers**

**Problem:** Write a function to find the **maximum** of two numbers.

**📌 2. Pass by Value vs. Pass by Reference: Swap Two Numbers**

**Problem:** Write a function that swaps two numbers using **pass by value** and **pass by reference**.

**📌 3. Function Overloading: Calculate the Area**

**Problem:** Implement function overloading to calculate the **area of a circle, rectangle, and square**.

**📌 4. Recursion: Find Factorial of a Number**

**Problem:** Write a recursive function to find the **factorial** of a number.

**📌 5. Recursion: Print Fibonacci Series (N Terms)**

**Problem:** Implement a recursive function to print the **Fibonacci series** up to N terms.

**📌 6. Tail Recursion vs. Non-Tail Recursion: Sum of First N Natural Numbers**

**Problem:** Compare **tail recursion** and **non-tail recursion** to compute the sum of first N natural numbers.

**📌 7. Backtracking: Generate All Subsets of a Set**

**Problem:** Write a recursive function to generate all **subsets of a given set**.

**📌 8. Backtracking: Solve N-Queens Problem**

**Problem:** Implement the **N-Queens** problem using backtracking.

**📌 9. Backtracking: Sudoku Solver**

**Problem:** Solve a **Sudoku puzzle** using backtracking.

**📌 10. Recursion: Tower of Hanoi Problem**

**Problem:** Implement the **Tower of Hanoi** problem using recursion.­­

## 📌 1. Function Basics: Find the Maximum of Two Numbers

**Problem:** Write a function to find the maximum of two numbers.

### C++

### #include <iostream>

using namespace std;

int maxNumber(int a, int b) {

return (a > b) ? a : b;

}

int main() {

int x, y; cin >> x >> y;

cout << maxNumber(x, y);

return 0;

}

### Java

import java.util.Scanner;

class Main {

static int maxNumber(int a, int b) {

return (a > b) ? a : b;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int x = sc.nextInt(), y = sc.nextInt();

System.out.println(maxNumber(x, y));

}

}

### Python

def max\_number(a, b):

return max(a, b)

x, y = map(int, input().split())

print(max\_number(x, y))

## 📌 2. Pass by Value vs. Pass by Reference: Swap Two Numbers

**Problem:** Write a function that swaps two numbers using pass by value and pass by reference.

### C++ (Pass by Reference)

#include <iostream>

using namespace std;

void swap(int &a, int &b) {

int temp = a;

a = b;

b = temp;

}

int main() {

int x, y; cin >> x >> y;

swap(x, y);

cout << x << " " << y;

return 0;

}

### Java (Using Array to Simulate Reference)

import java.util.Scanner;

class Main {

static void swap(int[] arr) {

int temp = arr[0];

arr[0] = arr[1];

arr[1] = temp;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int[] arr = {sc.nextInt(), sc.nextInt()};

swap(arr);

System.out.println(arr[0] + " " + arr[1]);

}

}

### Python (Using Tuple Swap)

def swap(a, b):

return b, a

x, y = map(int, input().split())

x, y = swap(x, y)

print(x, y)

## 📌 3. Function Overloading: Calculate the Area

**Problem:** Implement function overloading to calculate the area of a circle, rectangle, and square.

### C++

#include <iostream>

using namespace std;

double area(double r) { return 3.1415 \* r \* r; }

double area(double l, double w) { return l \* w; }

double area(double s) { return s \* s; }

int main() {

cout << area(5.0) << " " << area(4.0, 6.0) << " " << area(4.0);

return 0;

}

### Java

class Main {

static double area(double r) { return 3.1415 \* r \* r; }

static double area(double l, double w) { return l \* w; }

static double area(double s) { return s \* s; }

public static void main(String[] args) {

System.out.println(area(5.0) + " " + area(4.0, 6.0) + " " + area(4.0));

}

}

### Python

def area(r): return 3.1415 \* r \* r

def area(l, w): return l \* w

def area(s): return s \* s

print(area(5.0), area(4.0, 6.0), area(4.0))

## 📌 4. Recursion: Find Factorial of a Number

### C++

#include <iostream>

using namespace std;

long long factorial(int n) {

return (n == 0) ? 1 : n \* factorial(n - 1);

}

int main() {

int n; cin >> n;

cout << factorial(n);

return 0;

}

### Java

import java.util.Scanner;

class Main {

static long factorial(int n) {

return (n == 0) ? 1 : n \* factorial(n - 1);

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int n = sc.nextInt();

System.out.println(factorial(n));

}

}

### Python

def factorial(n):

return 1 if n == 0 else n \* factorial(n - 1)

n = int(input())

print(factorial(n))

## 📌 5. Recursion: Print Fibonacci Series (N Terms)

### C++

#include <iostream>

using namespace std;

void fibonacci(int n, int a = 0, int b = 1) {

if (n == 0) return;

cout << a << " ";

fibonacci(n - 1, b, a + b);

}

int main() {

int n; cin >> n;

fibonacci(n);

return 0;

}

### Java

import java.util.Scanner;

class Main {

static void fibonacci(int n, int a, int b) {

if (n == 0) return;

System.out.print(a + " ");

fibonacci(n - 1, b, a + b);

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int n = sc.nextInt();

fibonacci(n, 0, 1);

}

}

### Python

def fibonacci(n, a=0, b=1):

if n == 0:

return

print(a, end=" ")

fibonacci(n - 1, b, a + b)

n = int(input())

fibonacci(n)

## 📌 6. Tail Recursion vs. Non-Tail Recursion: Sum of First N Natural Numbers

### C++

#include <iostream>

using namespace std;

int sumTail(int n, int accumulator = 0) {

if (n == 0) return accumulator;

return sumTail(n - 1, accumulator + n);

}

int sumNonTail(int n) {

if (n == 0) return 0;

return n + sumNonTail(n - 1);

}

int main() {

int n; cin >> n;

cout << sumTail(n) << " " << sumNonTail(n);

return 0;

}

### Java

import java.util.Scanner;

class Main {

static int sumTail(int n, int accumulator) {

if (n == 0) return accumulator;

return sumTail(n - 1, accumulator + n);

}

static int sumNonTail(int n) {

if (n == 0) return 0;

return n + sumNonTail(n - 1);

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int n = sc.nextInt();

System.out.println(sumTail(n, 0) + " " + sumNonTail(n));

}

}

### Python

def sum\_tail(n, accumulator=0):

return accumulator if n == 0 else sum\_tail(n - 1, accumulator + n)

def sum\_non\_tail(n):

return 0 if n == 0 else n + sum\_non\_tail(n - 1)

n = int(input())

print(sum\_tail(n), sum\_non\_tail(n))

## 📌 7. Backtracking: Generate All Subsets of a Set

### C++

#include <iostream>

#include <vector>

using namespace std;

void generateSubsets(vector<int>& nums, vector<int>& subset, int index) {

if (index == nums.size()) {

for (int num : subset) cout << num << " ";

cout << endl;

return;

}

generateSubsets(nums, subset, index + 1);

subset.push\_back(nums[index]);

generateSubsets(nums, subset, index + 1);

subset.pop\_back();

}

int main() {

vector<int> nums = {1, 2, 3}, subset;

generateSubsets(nums, subset, 0);

return 0;

}

### Java

import java.util.\*;

class Main {

static void generateSubsets(List<Integer> nums, List<Integer> subset, int index) {

if (index == nums.size()) {

System.out.println(subset);

return;

}

generateSubsets(nums, subset, index + 1);

subset.add(nums.get(index));

generateSubsets(nums, subset, index + 1);

subset.remove(subset.size() - 1);

}

public static void main(String[] args) {

List<Integer> nums = Arrays.asList(1, 2, 3);

generateSubsets(nums, new ArrayList<>(), 0);

}

}

### Python

def generate\_subsets(nums, subset, index):

if index == len(nums):

print(subset)

return

generate\_subsets(nums, subset, index + 1)

generate\_subsets(nums, subset + [nums[index]], index + 1)

numbers = [1, 2, 3]

generate\_subsets(numbers, [], 0)

## 📌 8. Backtracking: Solve N-Queens Problem

### C++

#include <iostream>

#include <vector>

using namespace std;

void printSolution(vector<vector<int>>& board, int N) {

for (auto row : board) {

for (int cell : row) cout << (cell ? "Q " : "\_ ");

cout << endl;

}

cout << endl;

}

bool isSafe(vector<vector<int>>& board, int row, int col, int N) {

for (int i = 0; i < row; i++) if (board[i][col]) return false;

for (int i = row, j = col; i >= 0 && j >= 0; i--, j--) if (board[i][j]) return false;

for (int i = row, j = col; i >= 0 && j < N; i--, j++) if (board[i][j]) return false;

return true;

}

bool solveNQueens(vector<vector<int>>& board, int row, int N) {

if (row == N) { printSolution(board, N); return true; }

bool res = false;

for (int col = 0; col < N; col++) {

if (isSafe(board, row, col, N)) {

board[row][col] = 1;

res = solveNQueens(board, row + 1, N) || res;

board[row][col] = 0;

}

}

return res;

}

int main() {

int N = 4;

vector<vector<int>> board(N, vector<int>(N, 0));

solveNQueens(board, 0, N);

return 0;

}

### Java

import java.util.\*;

class NQueens {

static final int N = 8;

static boolean isSafe(int board[][], int row, int col) {

for (int i = 0; i < col; i++)

if (board[row][i] == 1) return false;

for (int i = row, j = col; i >= 0 && j >= 0; i--, j--)

if (board[i][j] == 1) return false;

for (int i = row, j = col; i < N && j >= 0; i++, j--)

if (board[i][j] == 1) return false;

return true;

}

static boolean solveNQueens(int board[][], int col) {

if (col >= N) return true;

for (int i = 0; i < N; i++) {

if (isSafe(board, i, col)) {

board[i][col] = 1;

if (solveNQueens(board, col + 1)) return true;

board[i][col] = 0;

}

}

return false;

}

public static void main(String args[]) {

int board[][] = new int[N][N];

solveNQueens(board, 0);

for (int[] row : board) {

for (int cell : row) System.out.print(cell + " ");

System.out.println();

}

}

}

### Python

def is\_safe(board, row, col, N):

for i in range(col):

if board[row][i] == 1:

return False

for i, j in zip(range(row, -1, -1), range(col, -1, -1)):

if board[i][j] == 1:

return False

for i, j in zip(range(row, N, 1), range(col, -1, -1)):

if board[i][j] == 1:

return False

return True

def solve\_nqueens(board, col, N):

if col >= N:

return True

for i in range(N):

if is\_safe(board, i, col, N):

board[i][col] = 1

if solve\_nqueens(board, col + 1, N):

return True

board[i][col] = 0

return False

N = 8

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

solve\_nqueens(board, 0, N)

for row in board:

print(" ".join(map(str, row)))

## 📌 9. Backtracking: Sudoku Solver

**Problem:** Solve a Sudoku puzzle using backtracking.

### C++

#include <iostream>

using namespace std;

#define N 9

bool isSafe(int grid[N][N], int row, int col, int num) {

for (int x = 0; x < N; x++)

if (grid[row][x] == num || grid[x][col] == num)

return false;

int startRow = row - row % 3, startCol = col - col % 3;

for (int i = 0; i < 3; i++)

for (int j = 0; j < 3; j++)

if (grid[i + startRow][j + startCol] == num)

return false;

return true;

}

bool solveSudoku(int grid[N][N], int row, int col) {

if (row == N - 1 && col == N) return true;

if (col == N) { row++; col = 0; }

if (grid[row][col] != 0) return solveSudoku(grid, row, col + 1);

for (int num = 1; num <= 9; num++) {

if (isSafe(grid, row, col, num)) {

grid[row][col] = num;

if (solveSudoku(grid, row, col + 1)) return true;

grid[row][col] = 0;

}

}

return false;

}

### Java

class SudokuSolver {

private static final int N = 9;

static boolean isSafe(int[][] board, int row, int col, int num) {

for (int x = 0; x < N; x++)

if (board[row][x] == num || board[x][col] == num)

return false;

int startRow = row - row % 3, startCol = col - col % 3;

for (int i = 0; i < 3; i++)

for (int j = 0; j < 3; j++)

if (board[i + startRow][j + startCol] == num)

return false;

return true;

}

static boolean solveSudoku(int[][] board, int row, int col) {

if (row == N - 1 && col == N) return true;

if (col == N) { row++; col = 0; }

if (board[row][col] != 0) return solveSudoku(board, row, col + 1);

for (int num = 1; num <= 9; num++) {

if (isSafe(board, row, col, num)) {

board[row][col] = num;

if (solveSudoku(board, row, col + 1)) return true;

board[row][col] = 0;

}

}

return false;

}

}

### Python

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

for x in range(9):

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

return False

start\_row, start\_col = row - row % 3, col - col % 3

for i in range(3):

for j in range(3):

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

return False

return True

def solve\_sudoku(board, row=0, col=0):

if row == 8 and col == 9:

return True

if col == 9:

row += 1

col = 0

if board[row][col] != 0:

return solve\_sudoku(board, row, col + 1)

for num in range(1, 10):

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

board[row][col] = num

if solve\_sudoku(board, row, col + 1):

return True

board[row][col] = 0

return False

## 📌 10. Recursion: Tower of Hanoi Problem

**Problem:** Implement the Tower of Hanoi problem using recursion.

### C++

#include <iostream>

using namespace std;

void towerOfHanoi(int n, char from, char to, char aux) {

if (n == 1) {

cout << "Move disk 1 from " << from << " to " << to << endl;

return;

}

towerOfHanoi(n - 1, from, aux, to);

cout << "Move disk " << n << " from " << from << " to " << to << endl;

towerOfHanoi(n - 1, aux, to, from);

}

int main() {

int n; cin >> n;

towerOfHanoi(n, 'A', 'C', 'B');

return 0;

}

### Java

class TowerOfHanoi {

static void towerOfHanoi(int n, char from, char to, char aux) {

if (n == 1) {

System.out.println("Move disk 1 from " + from + " to " + to);

return;

}

towerOfHanoi(n - 1, from, aux, to);

System.out.println("Move disk " + n + " from " + from + " to " + to);

towerOfHanoi(n - 1, aux, to, from);

}

public static void main(String[] args) {

int n = 3;

towerOfHanoi(n, 'A', 'C', 'B');

}

}

### Python

def tower\_of\_hanoi(n, from\_rod, to\_rod, aux\_rod):

if n == 1:

print(f"Move disk 1 from {from\_rod} to {to\_rod}")

return

tower\_of\_hanoi(n - 1, from\_rod, aux\_rod, to\_rod)

print(f"Move disk {n} from {from\_rod} to {to\_rod}")

tower\_of\_hanoi(n - 1, aux\_rod, to\_rod, from\_rod)

🎉 **Congratulations!** If you've completed all these questions, you've taken a huge step in mastering functions, recursion, and backtracking. Keep practicing and challenging yourself! 🚀

📢 Stay connected for more coding content and problem-solving strategies! Follow me on Instagram: **@SYNTAX\_ERROR**