Session 5: Functions & Recursion

focusing on modular code design through reusable blocks and self-referencing logic to solve complex problems efficiently.

TeachToTech

What is a Function?

A function is a self-contained block of code designed to perform a specific task.
Instead of repeating code, we give the task a name and reuse it.

What is a Function?

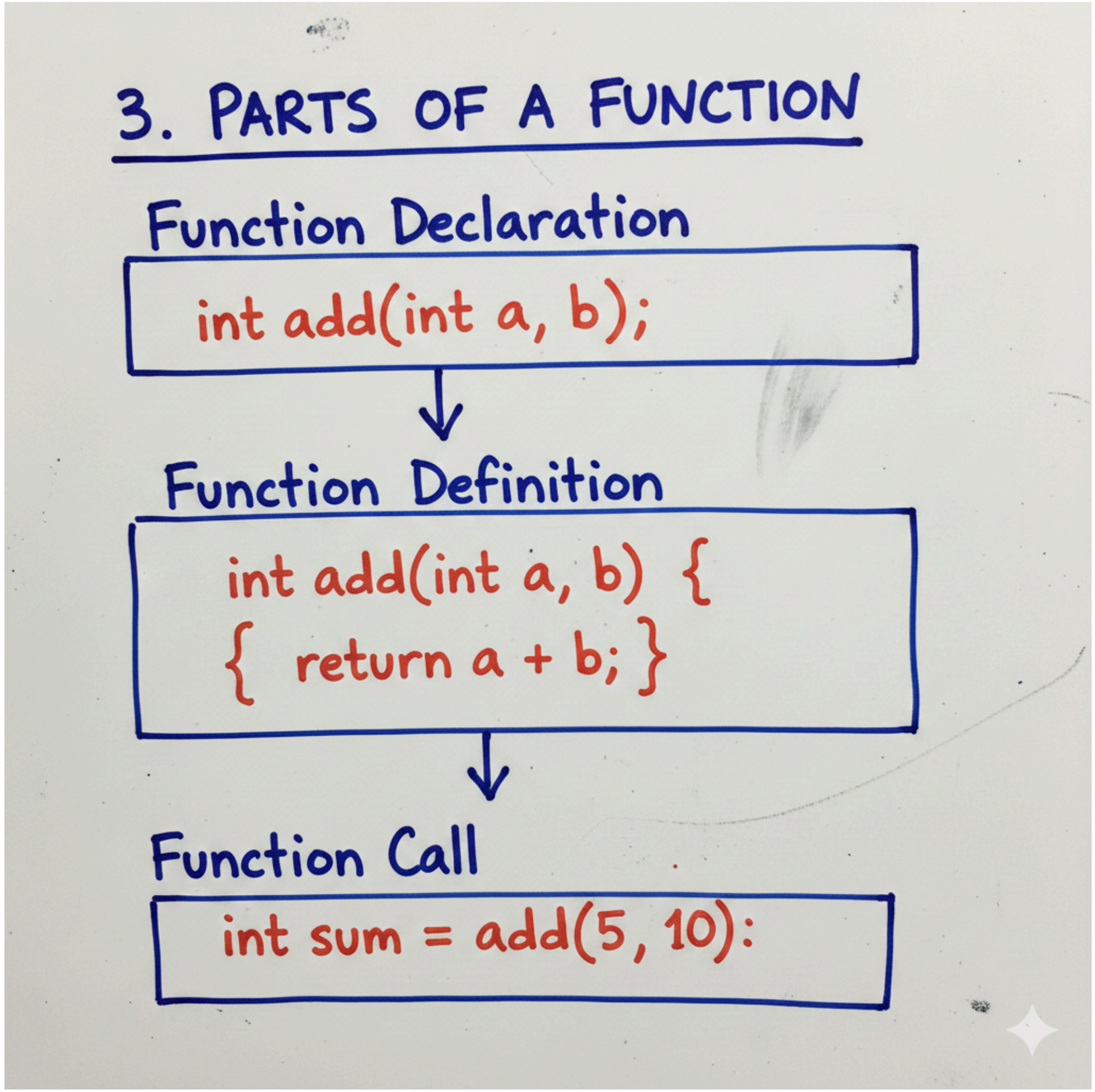
Every morning, your father says:
“Chai bana do.”
You don’t ask kaise?
You already know the steps — gas on, paani, chai patti, doodh, boil, serve.
That entire routine is a function.

Why Functions are Important

- Code reusability
- Better readability
- Easy debugging
- Structured programming

Real-world analogy:
One recipe, cooked many times.

Parts of a Function



Parts of a Function

Let’s Code

C_Program_Example.c

```c
#include <stdio.h>

// 1. Declaration (The Heads Up)
void greetElder(char name[]);

int main() {
    // 3. Function Call (The Action)
    // You meet your Uncle (Chacha Ji) and perform the action
    greetElder("Chacha Ji");

    return 0;
}

// 2. Definition (The Actual Logic)
void greetElder(char name[]) {
    printf("Namaste %s, kaise hain aap? (Touching feet...)\n", name);
}
```

8. Pass by Value

Copies value to function

Original value does not change

```
void change(int x) {
 x = 10;
}
```

Pass by Value.c

```c
#include <stdio.h>

// This function takes an integer by VALUE
void change(int x) {
    x = 10; // Only the local 'photocopy' inside this function changes
    printf("Inside change function, x = %d\n", x);
}

int main() {
    int myNumber = 5; // The "Original Document"

    printf("Before calling change: %d\n", myNumber);

    // We pass the value (5) to the function
    change(myNumber);

    // The original value remains 5 because the function worked on a copy
    printf("After calling change: %d\n", myNumber);

    return 0;
}
```

9. Pass by Reference

Uses address

Original value changes

```
void change(int *x) {
 *x = 10;
}
```

Pass by Reference.c

```c
#include <stdio.h>

// This function takes a pointer to an integer (the address of the
variable)
void change(int *x) {
    *x = 10; // Dereferencing: Go to that address and change the value
to 10
}

int main() {
    int myNumber = 5;

    printf("Before calling change: %d\n", myNumber);

    // We pass the address of myNumber using the '&' operator
    change(&myNumber);

    // The value has been changed in the original memory location
    printf("After calling change: %d\n", myNumber);

    return 0;
}
```

10. What is Recursion?

A function calling itself to solve a problem.

Key idea:

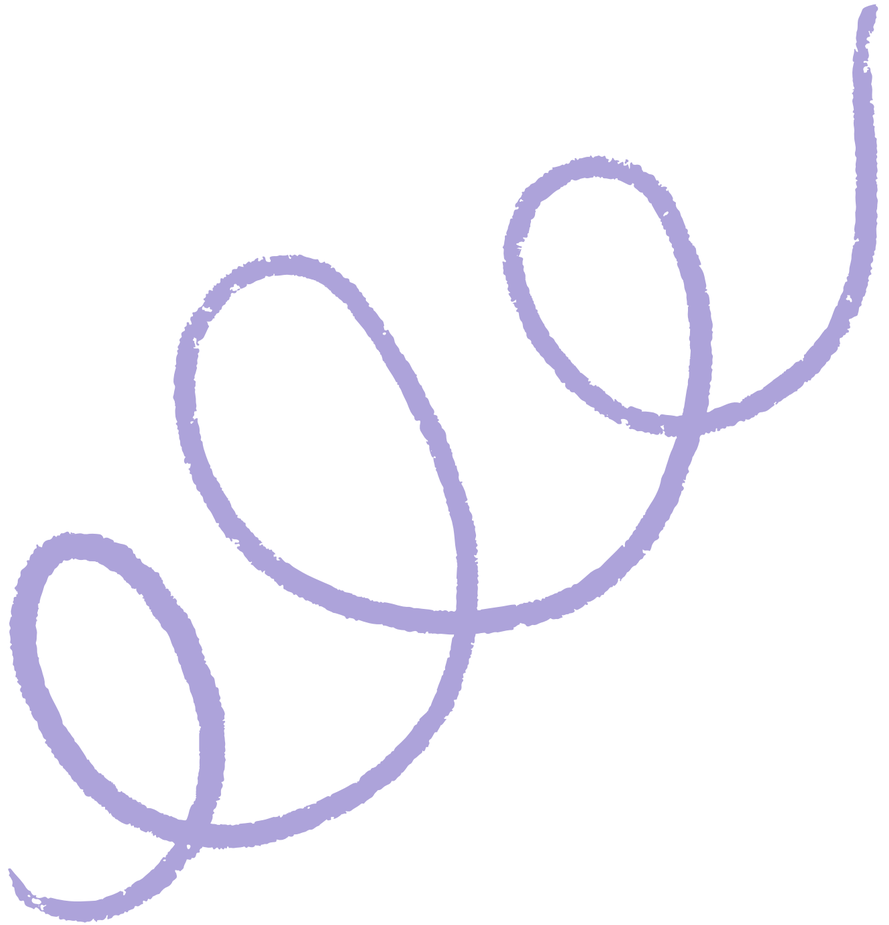
A big problem is broken into smaller versions of the same problem.

10. What is Recursion?

Peeling an Onion: You peel a layer to find another layer, until you reach the core (Base Case).

Components of Recursion

- Base Case – stops recursion
- Recursive Case – function calls itself



Without base case → infinite loop.

Recursion Example: Print 1 to N

```
void print(int n) {
 if(n == 0)
 return;
 print(n - 1);
 printf("%d ", n);
}
```

Recursion Example: Fibonacci Number

```
int fib(int n) {
    if(n <= 1)
        return n;
    return fib(n-1) + fib(n-2);
}
```

Recursion Example: Reverse String

```
void reverse(char str[], int i) {
    if(str[i] == '\0')
        return;
    reverse(str, i + 1);
    printf("%c", str[i]);
}
```

Activity: Factorial Using Recursion

```
int fact(int n) {
    if(n == 0)
        return 1;
    return n * fact(n - 1);
}
```

When to Use Recursion vs Loop

Recursion	**Loop**
Elegant logic	Faster
Uses stack	Uses less memory
Risk of stack overflow	Safer for large input

Session Summary

- Functions divide logic
- Pass by value vs reference
- Recursion simplifies complex problems
- Base case is critical