**Return means to return the control from where we are calling so in recursion previous functionl;**

**🔁 What happens when you use return in recursion?**

When you write:

c

CopyEdit

if (base\_condition)

return;

It **does not end the entire program or main function**, it only:

👉 **Ends the current function call** and  
👉 Returns control to **its caller** (the previous function call in the recursion chain).

**📚 Example**

c

CopyEdit

void printDecreasing(int n) {

if (n == 0)

return; // base case: stop here

printf("%d ", n); // print current

printDecreasing(n - 1); // recursive call

}

Now let’s say you call:

c

CopyEdit

printDecreasing(3);

The flow will be:

* printDecreasing(3) → prints 3 → calls printDecreasing(2)
* printDecreasing(2) → prints 2 → calls printDecreasing(1)
* printDecreasing(1) → prints 1 → calls printDecreasing(0)
* printDecreasing(0) → hits base case → returns to printDecreasing(1)
* ... and then each function finishes and returns up the chain.

🧠 So, return only ends the current call — not the entire recursion at once.

**⚠️ Common confusion:**

You may think:

“I used return in base case so the whole thing should stop!”

But no — it's like a ladder: every step has to return *one by one* back up, unless you return early from the *main calling function*, which usually isn't the case in recursion.

**🔄 What if you return a value?**

c

CopyEdit

int factorial(int n) {

if (n == 0) return 1;

return n \* factorial(n - 1);

}

Here, each call **waits** for the result of the next call, and the value goes back up like:

scss

CopyEdit

factorial(3)

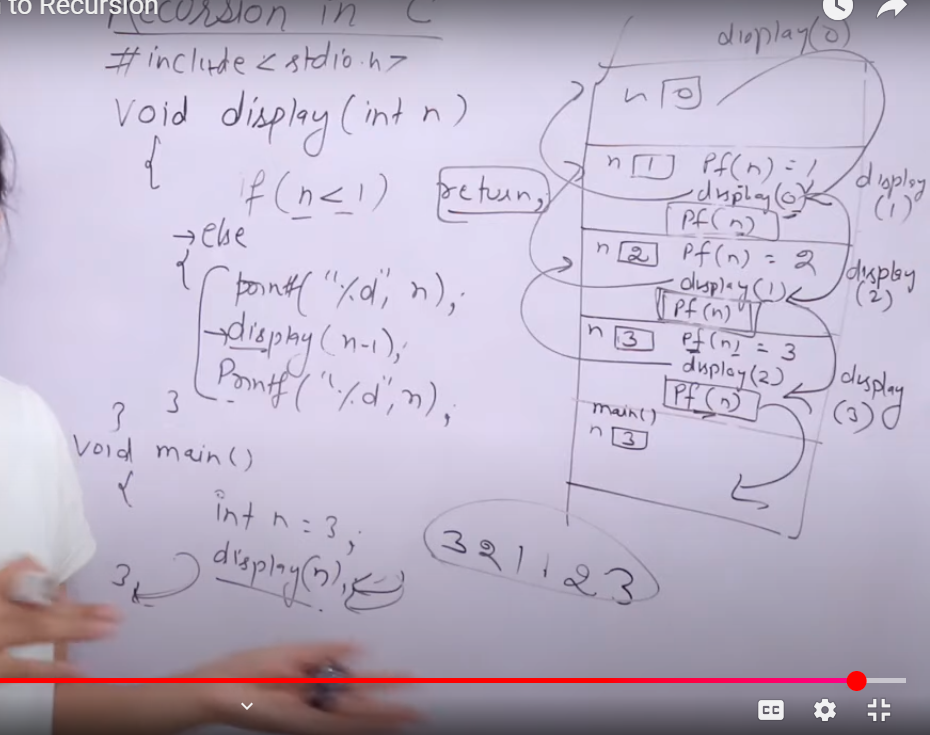
→ 3 \* factorial(2)

→ 3 \* (2 \* factorial(1))

→ 3 \* (2 \* (1 \* factorial(0)))

→ 3 \* (2 \* (1 \* 1))

→ 6



### 🔄 What is Recursion?

Recursion is when a function **calls itself** to solve a **smaller sub-problem**. Each function call is kept in memory using a **stack** (LIFO — Last In, First Out).

### 🧠 Important Concepts:

1. **Every time a function is called**, a new **stack frame (activation record)** is created for it.
2. Each stack frame contains:
   * Local variables
   * Return address
   * Parameters passed
3. When base case is met, the recursion stops **going deeper**, and the function starts to **return** one by one.

### 📊 Stack Trace and Flow (Step-by-step):

#### Step 1: main() is called.

* Local variable: n = 3
* Calls display(3)

### ➡️ First Call: display(3)

* Stack frame for display(3)
* n = 3
* Checks: 3 < 1 ❌
* Prints: 3
* Calls: display(2)

### ➡️ Second Call: display(2)

* Stack frame for display(2)
* n = 2
* Checks: 2 < 1 ❌
* Prints: 2
* Calls: display(1)

### ➡️ Third Call: display(1)

* Stack frame for display(1)
* n = 1
* Checks: 1 < 1 ❌
* Prints: 1
* Calls: display(0)

### ➡️ Fourth Call: display(0)

* Stack frame for display(0)
* n = 0
* Checks: 0 < 1 ✅
* **Hits base case** → returns without printing anything.

### 🔙 Returning Back: Unwinding Phase

#### Now the stack unwinds:

1. **Back to display(1)**
   * After recursive call, executes: printf("%d", n); → Prints: 1
   * Returns to display(2)
2. **Back to display(2)**
   * Executes: printf("%d", n); → Prints: 2
   * Returns to display(3)
3. **Back to display(3)**
   * Executes: printf("%d", n); → Prints: 3
   * Returns to main()

### ✅ Final Output:

CopyEdit

3 2 1 1 2 3

### 📌 Visualization (Stack Frames):

scss

CopyEdit

main()

└── display(3)

└── display(2)

└── display(1)

└── display(0) → return

← return → printf 1

← return → printf 2

← return → printf 3