Your Ultimate Guide To Landing Top AI roles

2.3.5

Introduction to Recursion

DECODE AiML

* What is Recursion?

→ Recursion means ① A function calling itself

② Function calling itself to solve a smaller part of the same problem.

Recursive Approach

```
def summ(n):
    if n==0:
        return 0
    return n+ summ(n-1)
```

Iterative Approach

```
def summ(n):
    total_sum=0
    for i in range(1,n+1):
        total_sum += i
    return total_sum
```

Ex 1: Sum of first 5 natural numbers

1	2	3	4	5

Summ(5) = 1+ 2+ 3 + 4 + 5

→ Let's build solution bottom up

$Summ(1) = 1 + 0 = 1 + Summ(0)$

$Summ(2) = 2 + 1 = 2 + Summ(1)$

$Summ(3) = 3 + 2 + 1 = 3 + Summ(2)$

$Summ(4) = 4 + 3 + 2 + 1 = 4 + Summ(3)$

$Summ(0) = 0$

↳ Base Case

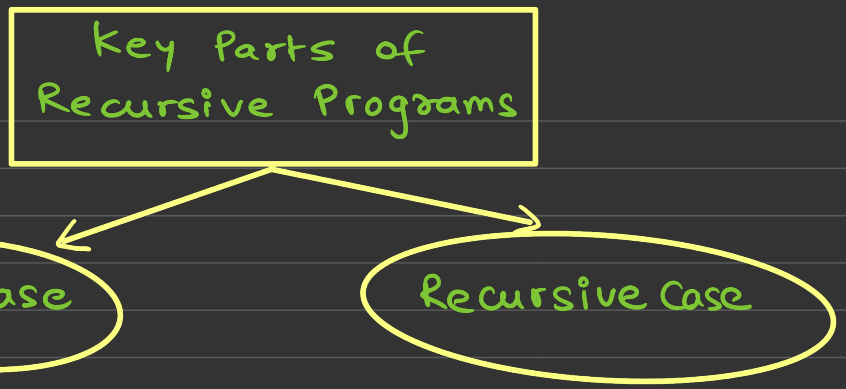
$$Summ(n) = n + Summ(n-1)$$

↳ Recursive equation

→ A Recursive equation is a mathematical formula that defines each term in a sequence based on Previous term.

→ To write recursive programs, you need

1. Base Case (Trivial case)
2. Recursive equation



DECODE AiML

Recursive Approach

```
def summ(n):
    if n==0:
        return 0
    return n+ summ(n-1)
```

Base Case

Recursive Case

Ex 2: Factorial of a number

n or $n!$ ← Representation

$$fact(n) = n \times (n-1) \times (n-2) \times \cdots \times 1$$

↳ Let's build solution bottom up.

$$fact(1) = 1 = 1 \times fact(0)$$

$$fact(2) = 2 \times 1 = 2 \times fact(1)$$

$$fact(3) = 3 \times 2 \times 1 = 3 \times fact(2)$$

$$fact(4) = 4 \times 3 \times 2 \times 1 = 4 \times fact(3)$$

$$fact(n) = n \times fact(n-1)$$ ← Recursive equation.

$$fact(0) = 1$$

↑ Base Case or Trivial Case

↑ Smallest subproblem which can't be further broken down.

↳ Let's write the recursive program to calculate the factorial of a number n.

Recursive Approach

```
def fact(n):
    if n==0:
        return 1
    return n x fact(n-1)
```

→ Base Case

→ Recursive Case

Ex: Find the n^{th} fibonacci no.

* Fibonacci sequence

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

↳ Fibonacci isn't a normal sequence. Lots of usecases

① DP → Counting ways to climb stairs

② Financial market → Technical Analysis (Support/Resistant)

③ Nature & Biology → Flower petals (often 3, 5, 8, 13 ---)

④ Architecture & design → Golden ratio

⑤ Data structures → Fibonacci heaps

⑥ Music

↳ Approach

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ----

$fib(2) = 1 = 1 + 0 = fib(1) + fib(0)$
$fib(3) = 2 = 1 + 1 = fib(2) + fib(1)$
$fib(4) = 3 = 2 + 1 = fib(3) + fib(2)$
$fib(5) = 5 = 3 + 2 = fib(4) + fib(3)$
$fib(6) = 8 = 5 + 3 = fib(5) + fib(4)$

$fib(0) = 0$

$fib(1) = 1$

↑ Base case

$fib(n) = fib(n-1) + fib(n-2)$

↑ Recursive equation

Recursive Approach

```
def fib(n):
    if n==0 or n==1:
        return n
    return fib(n-1) + fib(n-2)
```

Base Case

Recursive Case

→ The above functions calculates n^{th} fibonacci number of a sequence

→ All recursive programs can be rewritten iteratively using stack.

→ But not all iterative programs are naturally or efficiently expressed Recursively.

Example: Sum of first 10^8 natural numbers.

Recursive Approach

```
def helper(n):
    if n==0:
        return 0
    return n+helper(n-1)
```

Iterative Approach

```
def helper(n):
    total_sum=0
    for i in range(1,n+1):
        total_sum += i
    return total_sum
```

→

```
Summ = helper(10000 0000)
print(f"sum is: {summ}")
```

NOTE:

① Most Programming language have a call stack size limit

② Deep recursion can cause a stack overflow.

③ some problems are naturally Iterative.

Ex: Read data from a DB and save it to a .csv file

→ All Programs can be solved Iteratively

* Does that means Iterative Approach is more powerful than Recursive Approach ?

↳ Recursion is great for elegance and Readability.

↳ Many Complex problems can be solved recursively in a few lines of code which will take multiple lines of code to implement using Iterative approach.

↳ ① Tree/Graph Traversal Algorithms
② Divide & Conquer Algorithms
③ Backtracking Algorithms
④ Dynamic Programming Algorithms

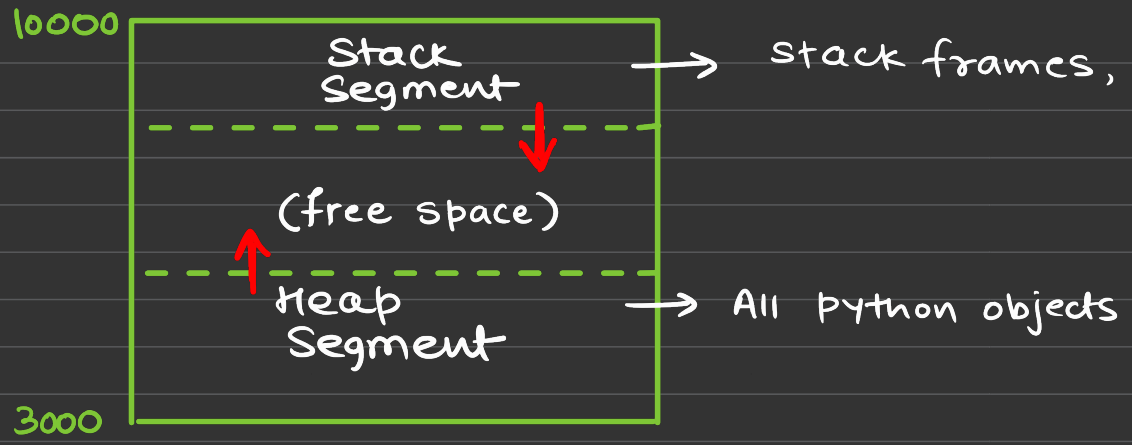
NOTE

1. Recursion is considered as the most trickiest topic to master
2. It is one of the main pillars of DSA. without Recursion you can't dream of Product based Companies.
3. Understanding Recursion will make you fall in love with Computer Science.
4. Most Important topic for DSA round of Interview.

* How Recursive Programs are handled in Main Memory?

↳ For a better understanding, do check Lecture

2.3.3 Memory Layout of Python Programs



→ For every Recursive function calls, a stack frame is allocated and deallocated whenever the function returns (completes it's execution).

→ Content of stack Frames

① References to local variables and Argument passed

② return address.

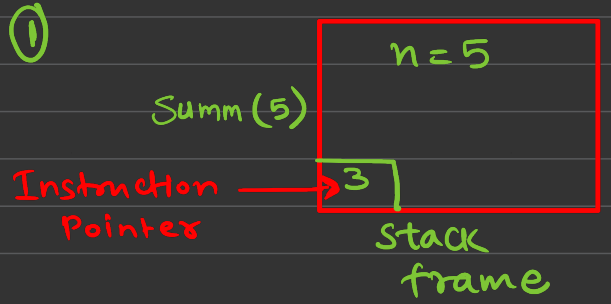
* Stack frames explained using Example

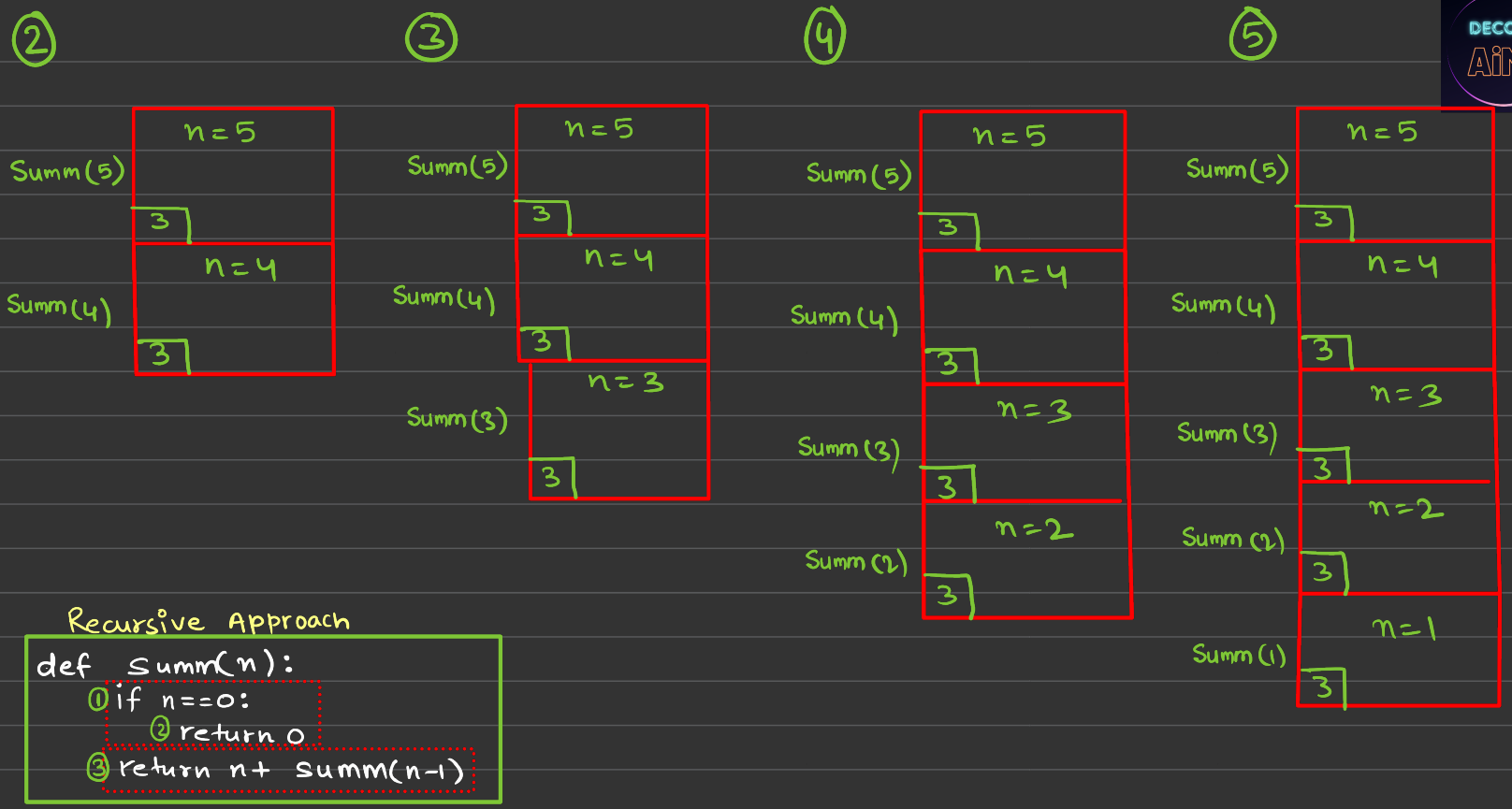
Example: Sum of first n natural numbers

Recursive Approach

```
def summ(n):
    ① if n==0:
        ② return 0
    ③ return n+ summ(n-1)
```

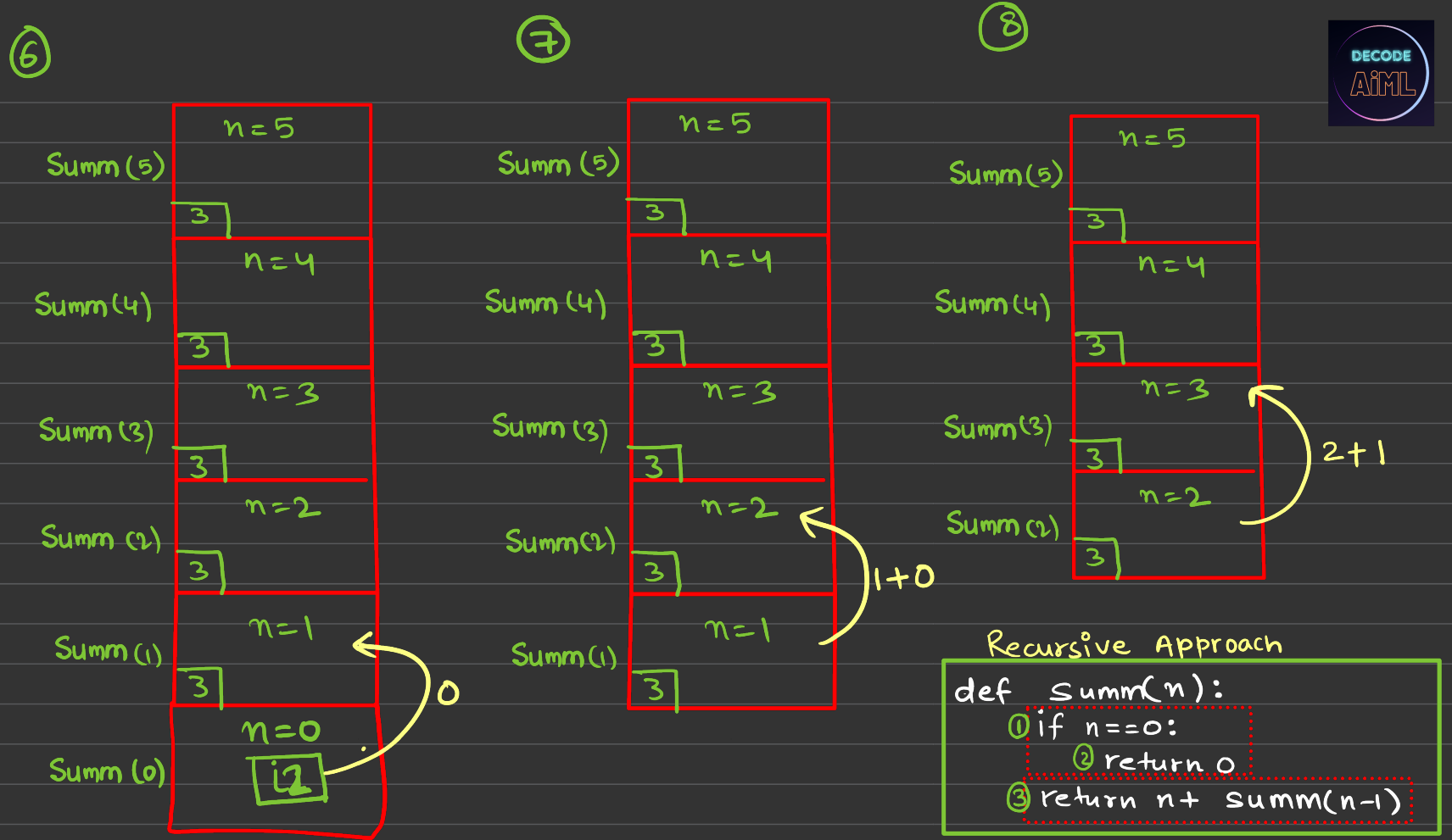
↳ Calculate Summ (5)



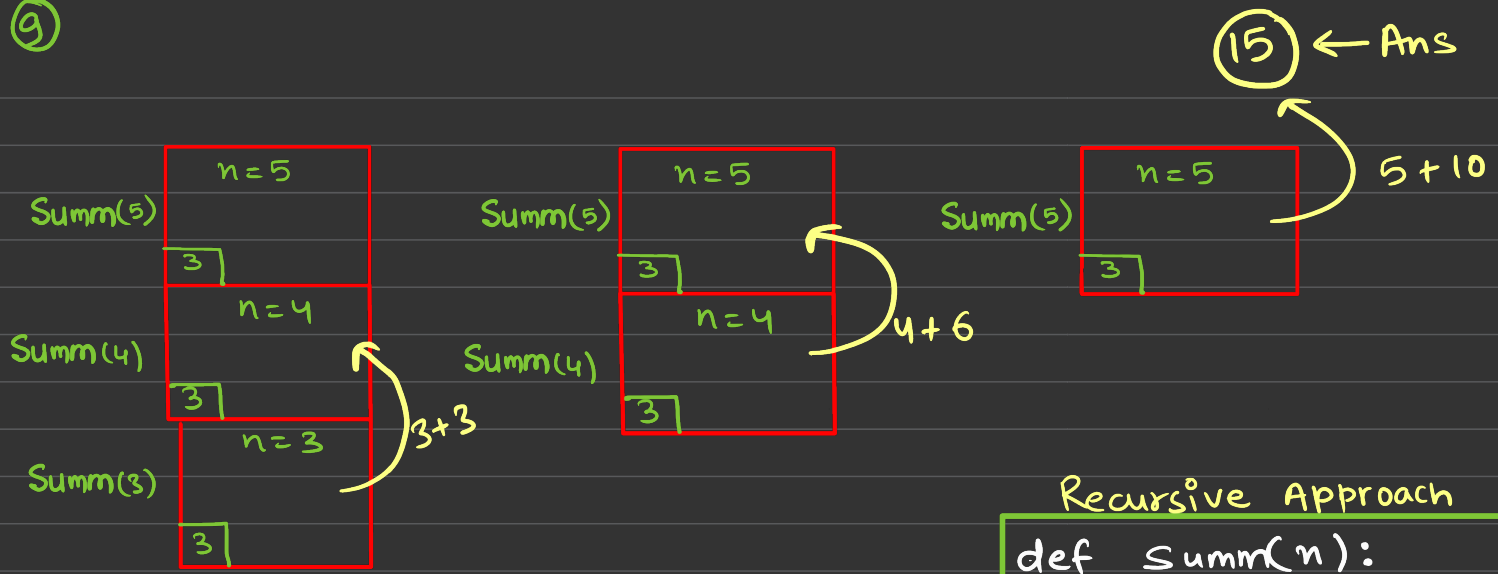


Recursive Approach

```
def summ(n):
    ① if n==0:
        ② return 0
    ③ return n+ summ(n-1)
```


6
7
8
DECODE AiML
n=5
Summ(5)
3
n=4
Summ(4)
3
n=3
Summ(3)
3
n=2
Summ(2)
3
n=1
Summ(1)
3
n=0
Summ(0)
i1
0
1+0
2+1
Recursive Approach
def summ(n):
① if n==0:
② return 0
③ return n+ summ(n-1)

9



Recursive Approach

```
def summ(n):
    ① if n==0:
        ② return 0
    ③ return n+ summ(n-1)
```

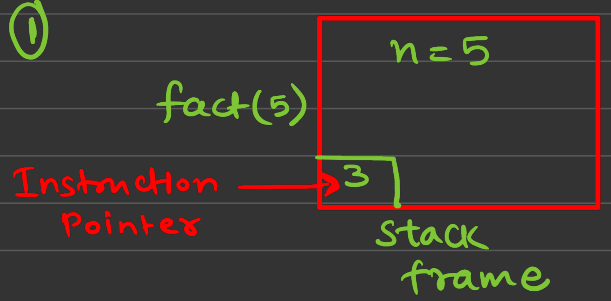
* Stack frames explained using Example

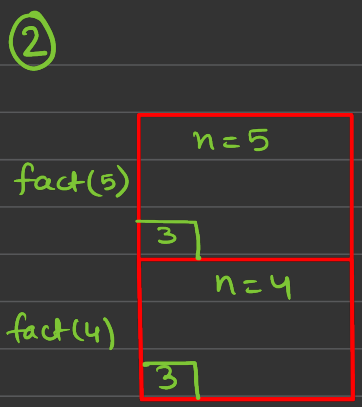
Example: factorial of a number n.

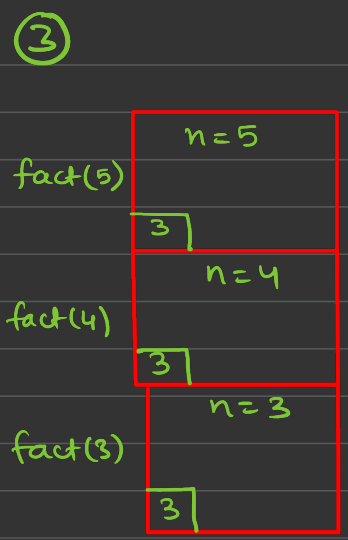
Recursive Approach

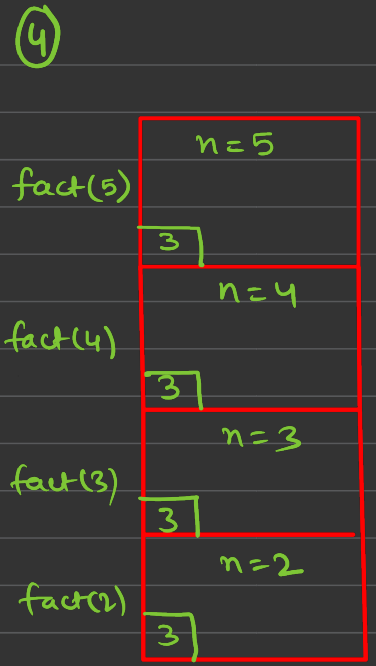
```
def fact(n):
    ① if n==0:
        ② return 1
    ③ return n x fact(n-1)
```

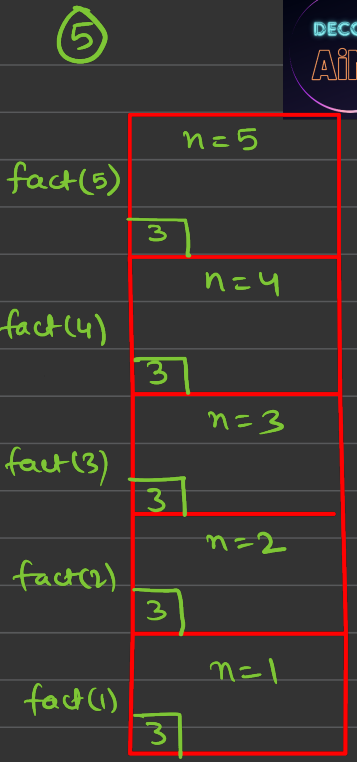
↳ Calculate fact(5)





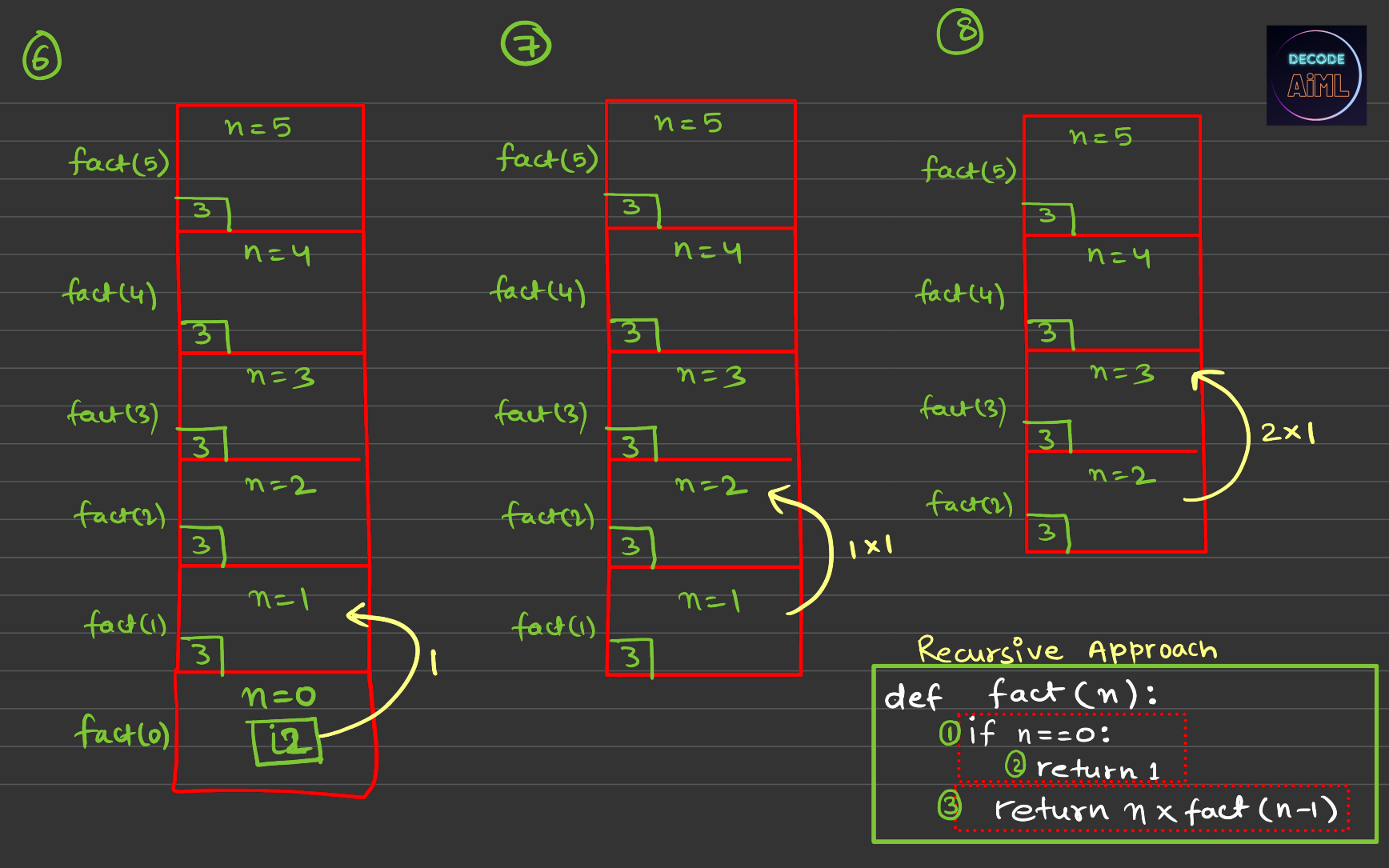




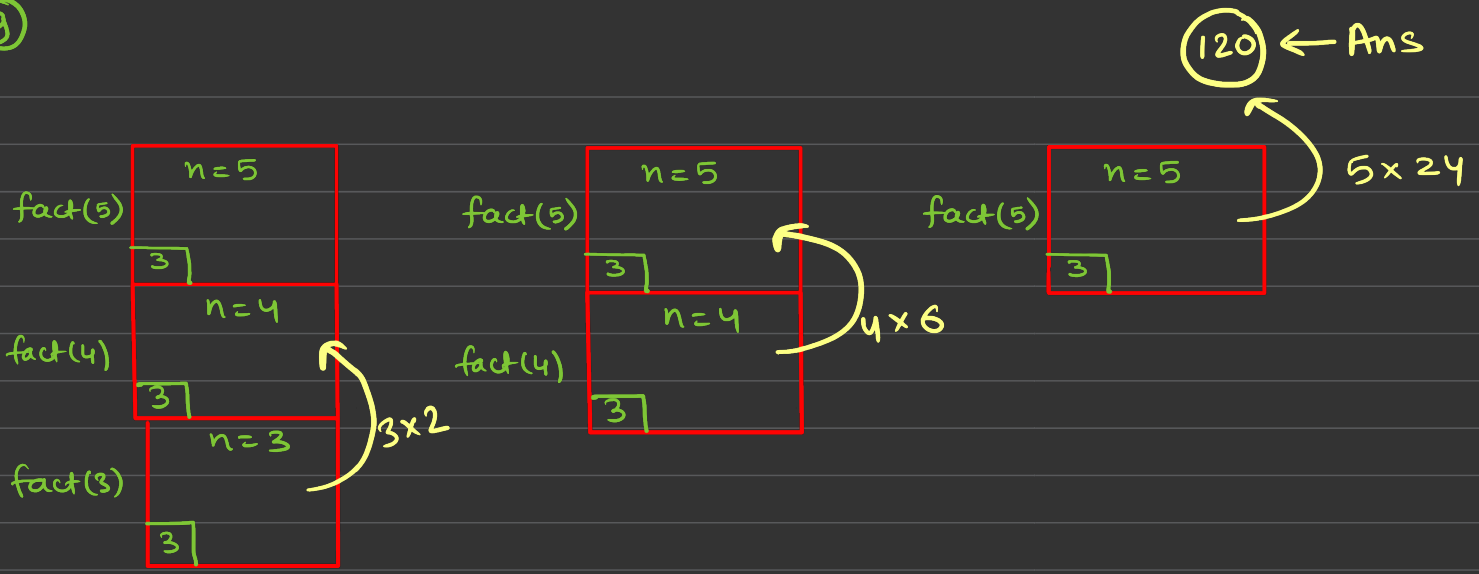


Recursive Approach

```
def fact(n):
    ① if n==0:
        ② return 1
    ③ return n x fact(n-1)
```

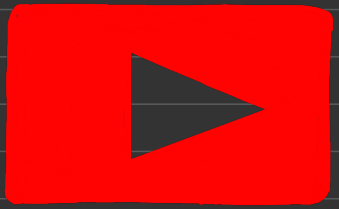

6
7
8
DECODE AiML
n=5
fact(5)
3
n=4
fact(4)
3
n=3
fact(3)
3
n=2
fact(2)
3
n=1
fact(1)
3
n=0
fact(0)
1
1
1x1
2x1
Recursive Approach
def fact(n):
① if n==0:
② return 1
③ return n x fact(n-1)

9



Recursive Approach

```
def fact(n):
    ① if n==0:
        ② return 1
    ③ return n x fact(n-1)
```

Like  Subscribe