Today's content.

- → Recursion
- → How to write recursive code / Tracing
- → tc / sc of recursive codes → Next class.

Why learn about recursion?

- → Merge sort / quick sort
- → Binary tree / BST / BBST / Segment tree / Tries
- → Dynamic programming
- → Backtracking
- → Graphs

Recursion

→ function calling itself.

→ Solving a problem, using smaller instance of same problem.
subproblem.

Sum(N) = 1+2+3+ --- + N-1 +N

Sum(N) = Sum(N-1) + N. → same functions of smaller size.

Sum(4) = Sum(3) + 4.

How to write recursive code?

Objective : what your fun should do.

Main logic : Solving objective using subproblem.

Base cond^n : Inputs for which we want to stop recursion.

Questions:

1)
```
def sum(N)      // Given N, calculate & return sum using recursion.
{
    if(N==0)
        return 1
    return sum(N-1)+N
}
```

Sum(11) = 1+2+3+--+11
= ( )
Sum(12) = 1+2+3+ --- +12
= sum(11) + 12.

Sum(N) = sum(N-1)+N.

2)
```
def factorial(N)
{   if(N==1) return 1;
    return N*factorial(N-1).
}
```

fact(3) = 3*2*1 , f(4) = 4*3*2*1

f(10) = 10*9*8* --- *2.

f(5) = 5*4*3*2*1 = 5*f(4).

stack trace → TODO.

f(8) = 8 * fact(7).

f(N) = N * fact(N-1).

Function call tracing.

```
def add(N,M)
    return N+M

def mul(x,y)
    return x*y

def sub(x,y)
    return x-y
```

```
main()
{
    x = 10, y = 20.
    print(sub(mul(add(x,y),30),75));
}
```

sub(mul(add(x,y),30),75))

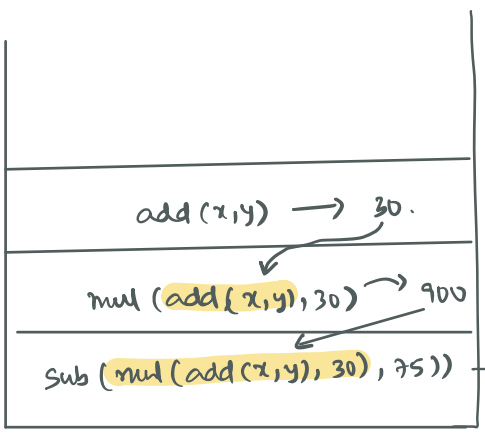
sub(mul(30,30),75))

sub(900,75)

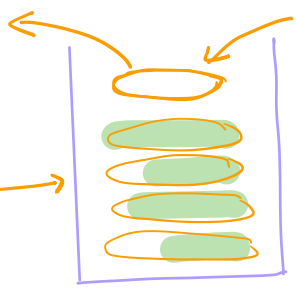
825.

1) When a function call happens, we add function call to the top.

2) When a function returns we remove it from top.

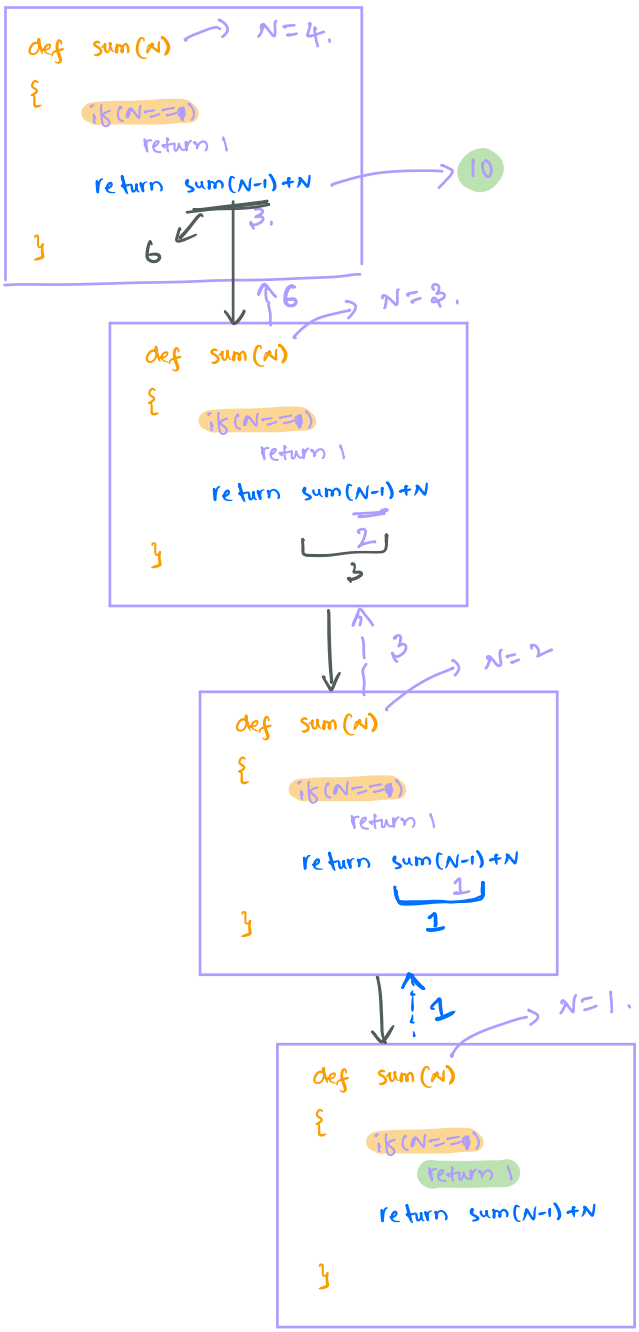


Data structure which we use to store function calls →

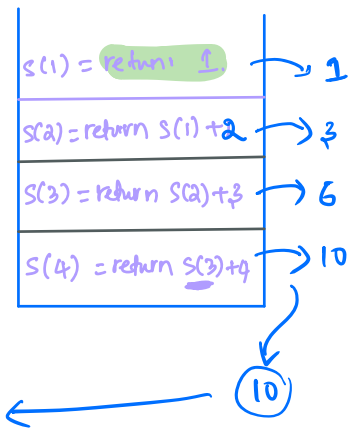


LIFO → Last in first out.

Stack of plates.



Stack trace.



Without base condition!: Recursion won't stop. [Memory limit error].

- TLE.
- Memory limit exceeded (Stack overflow).
  - Check for base condition.

Always write the base cond^n's at the start of the fu^n.

#input	0	1	2	3	4	5	6	7	8
Fib()	0	1	1	2	3	5	8	13	21.

f(10) →

```
def fib(N)  // Calculate & return Nth fibonacci number.
{
    if(n <= 1) return n
    return fib(n-1)+fib(n-2)
}
```

f(4) = f(3) + f(2).

f(8) = f(7) + f(6).

f(n) = f(n-1) + f(n-2)

How to calculate base case?.

f(2) = f(1) + f(0) =

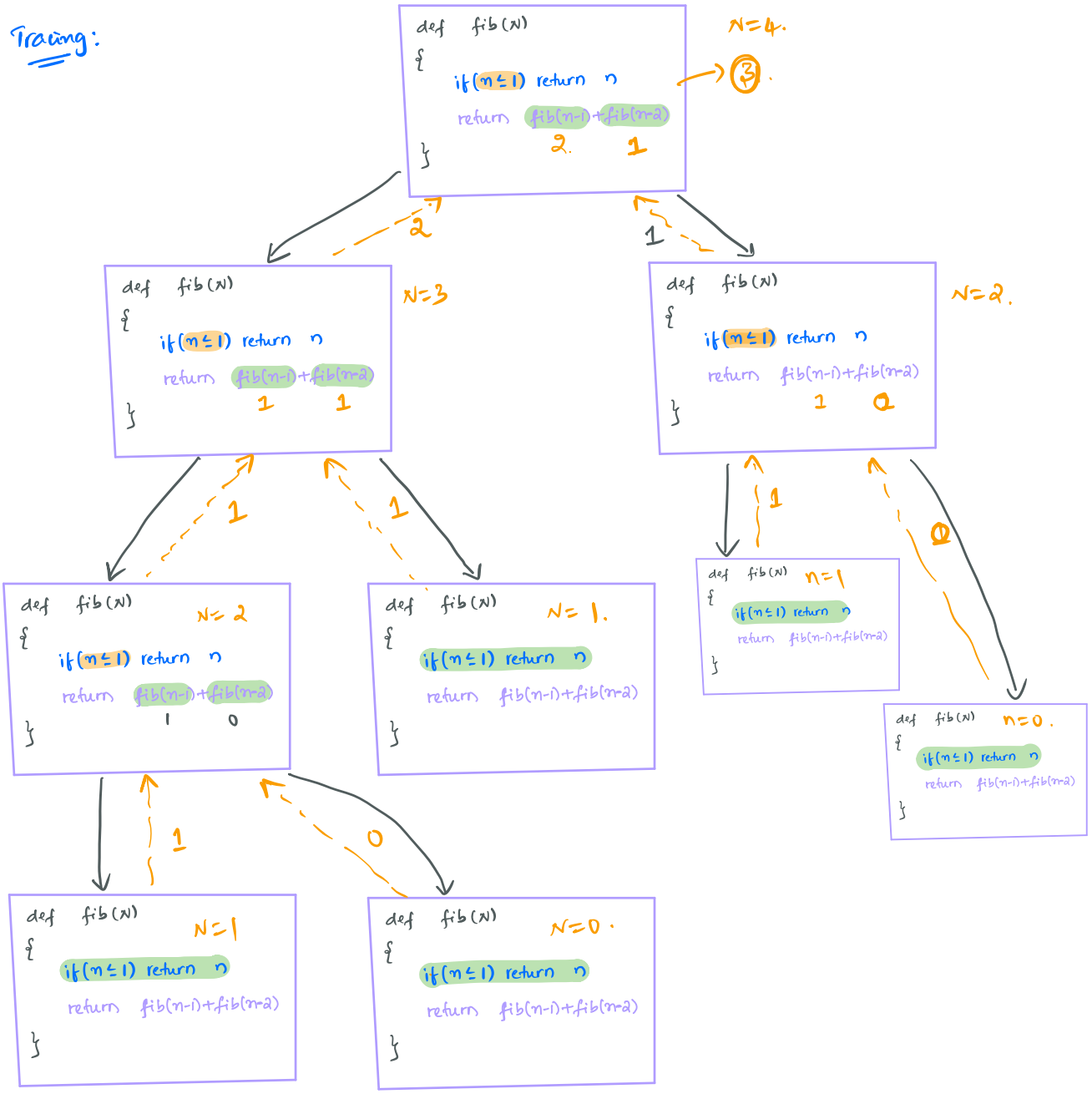
f(1) = f(0) + f(-1) → Problem.

f(0) = f(-1) + f(-2)

If n==1
    return 1
if n==0
    return 0.

→ if(n <= 1)
    return n.

Tracing:



Try tracing this with a stack. → TODO.

Break:

10:19:00

10:29:00.

Q: Given N, print all no's from 1 → N. (use recursion).

```
def increment(N)
{
    if(N==1)
        print 1
    else
        increment(N-1)
        print(N)
}
```

N = 4: 1, 2, 3, 4.

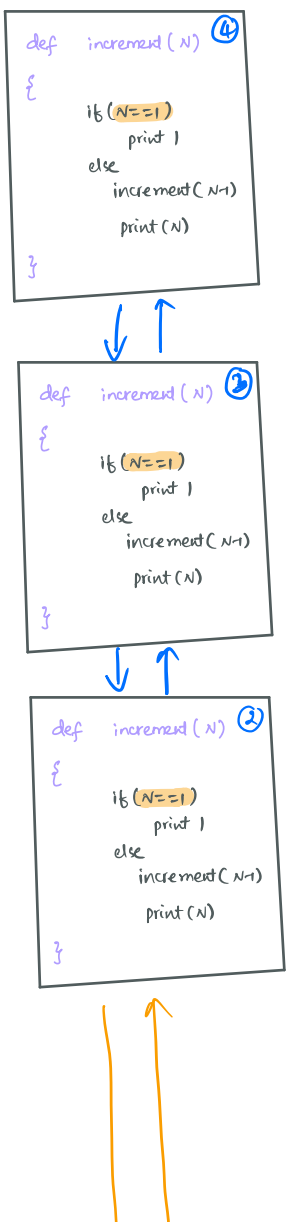
```
if(n==0)return
print(N).
increment(N-1).
```

4
3
2
1

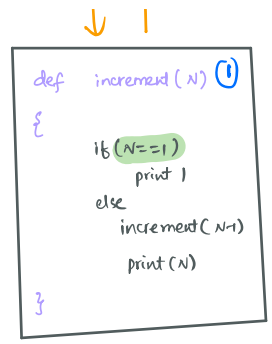
VL:

```
increment(N-1)
print(N).
```

1 ← f(1)
2 ← f(2)
3 ← f(3)
4 ← f(4)



print 1.
print 2
print 3
print 4



Try to print in decreasing order.

50) Given a substring, check whether its palindrome or not.

Ex: 0 1 2 3 4 5 6
goddaag s=4, e=6. ⟶ Yes.

s=2, e=5. ⟶ false.

```
def ispalindrome( word, s, e)
{
    if (s > e)
        return true
    if ( word[s] != word[e])
        return false.
    return ispalindrome(word, s+1, e-1)
}
```

0 1 2 3 4 5 6 7 8 9
h e l l o m a d a m

s=5, e=9.

5 6 7 8 9
m a d a m
— ~

s=s+1, e = e-1.

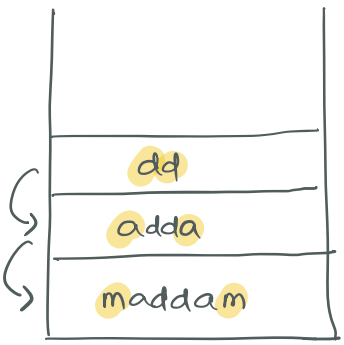
Tracing:

0 1 2 3 4 5

m a d d a m , s=0, e=5.

```
def ispalindrome( word, s, e)
{
    if (s >e)
        return true
    if ( word[s] != word[e])
        return false.
    return ispalindrome(word, s+1, e-1)
}
```

→ true.



true

```
def ispalindrome( word, s, e)
{
    if (s >e)
        return true
    if ( word[s] != word[e])
        return false.
    return ispalindrome(word, s+1, e-1)
}
```

1 → 4.

0 1 2 3 4 5

m a d d a m

true

```
def ispalindrome( word, s, e)
{
    if (s >e)
        return true
    if ( word[s] != word[e])
        return false.
    return ispalindrome(word, s+1, e-1)
}
```

2 → 3.

true

s = 3, e = 2.

```
def ispalindrome( word, s, e)
{
    if (s >e)
        return true
    if ( word[s] != word[e])
        return false.
    return ispalindrome(word, s+1, e-1)
}
```