Recursion - Part 1

May 4, 2022

AGENDA :

- How to write recursive code ?
- How does recursion work internally ?
- Some simple problems

Why Recursion ?

**

What is Recursion ?

→ a function calls itself

Why Recursion ?

- → Merge sort / Quicksort
- → Binary Tree / BST
- → Dynamic programming.
- → Graphs
- → Backtracking

Recursion

A function calling itself.

```
def sum( )
    - -
    - - -
```

* When you have to solve a large problem, use the function itself to solve a sub-problem.

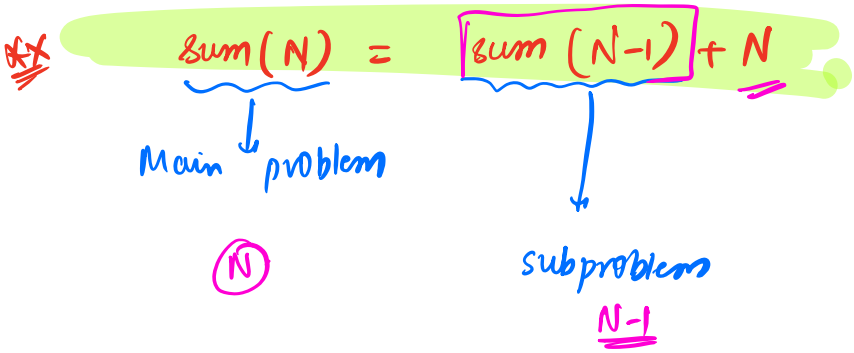
* Calculate sum of nos. from 1 to N.

```
def sum( N ) :
    s = 0
    for i in range(1, N+1):
        s = s+i

    return s
```

Iterative code (use for / while)

Problem **sum(N)** = $1 + 2 + 3 + 4 + 5 + 6 \ldots\ldots + N-1 + N$

$= \underbrace{1 + 2 + 3 + 4 + 5 + 6 \ldots\ldots + N-1}_{\text{sum}(N-1)} + N$



* How to write Recursive code?

1. **Assumption**: a) Define what your function is supposed to do.
   b) Believe in it; Believe that your function returns what you expect.

2. **Main Logic**: How do you solve the main problem in terms of sub-problem?

3. **Base condition**: What is the smallest sub-problem that your function can solve?

Q. Sum of nos from 1 to N.

1. Assumption:

```
def sum(N):
```

Relive that it returns sum of 1+2+3....N

sum(3) → 6
sum(2) → 3

2. Main Logic

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

Code:

```
return sum(N-1) + N
```

3. Base condition

```
if N == 1:
    return 1
```

Code:

```
def sum(N):
    if N == 1:
        return 1
    return N + sum(N-1)
```

Q. fact(N) = 1 * 2 * 3 * 4 * 5 . . . . N-1 * N

Write a code to compute factorial of N.

Recursive code:

1. Assumption:

```
def fact(N):
    Return 1*2*3...N
```

fact(3) → 6
fact(4) → 24

2. Main Logic

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

1 * 2 * 3 * 4 . . . . * N-1 * N

fact(N-1) * N

return N * fact(N-1)

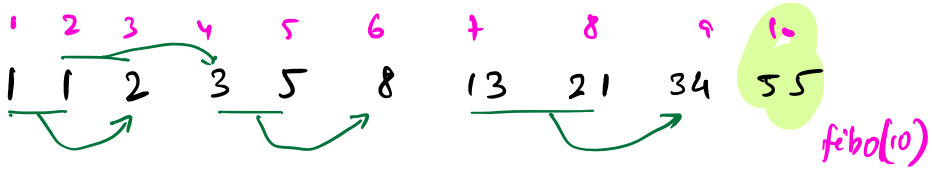
3. Base condition

```
if N == 1:
    return 1
```

Code:

```
def fact(N):
    if N==1:
        return 1
    return N * fact(N-1)
```

Q. Fibonacci series.



Golden ratio $\frac{55}{34}$ $\frac{34}{21}$ $\frac{21}{13}$

$\parallel$

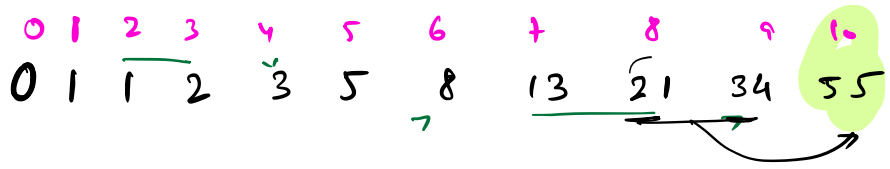
1.618. ϕ

Recursive code → To return Nth Fibonacci number.

Assumption:

```
def fibo(N):
    ↳ Return Nth fibonacci number
```

Main Logic:



fibo(10) = 34 + 21
= fibo(9) + fibo(8)

fibo(N) = fibo(N-1) + fibo(N-2)

return fibo(N-1) + fibo(N-2)

* Base condition:

if N==1 :
return 1 return 0
if N==2 :
return 1

fibo(3) = fibo(2) + fibo(1)
↑ valid ↑ inputs

** fibo(1) = fibo(0) + fibo(-1)
↑ Invalid ↑ Invalid

** fibo(2) = fibo(1) + fibo(0)
↑ valid ↑ Invalid

def fibo(N):

if N==1 OR N==2 :
return 1

return fibo(N-1) + fibo(N-2)

** Internals of Recursion

e.g.

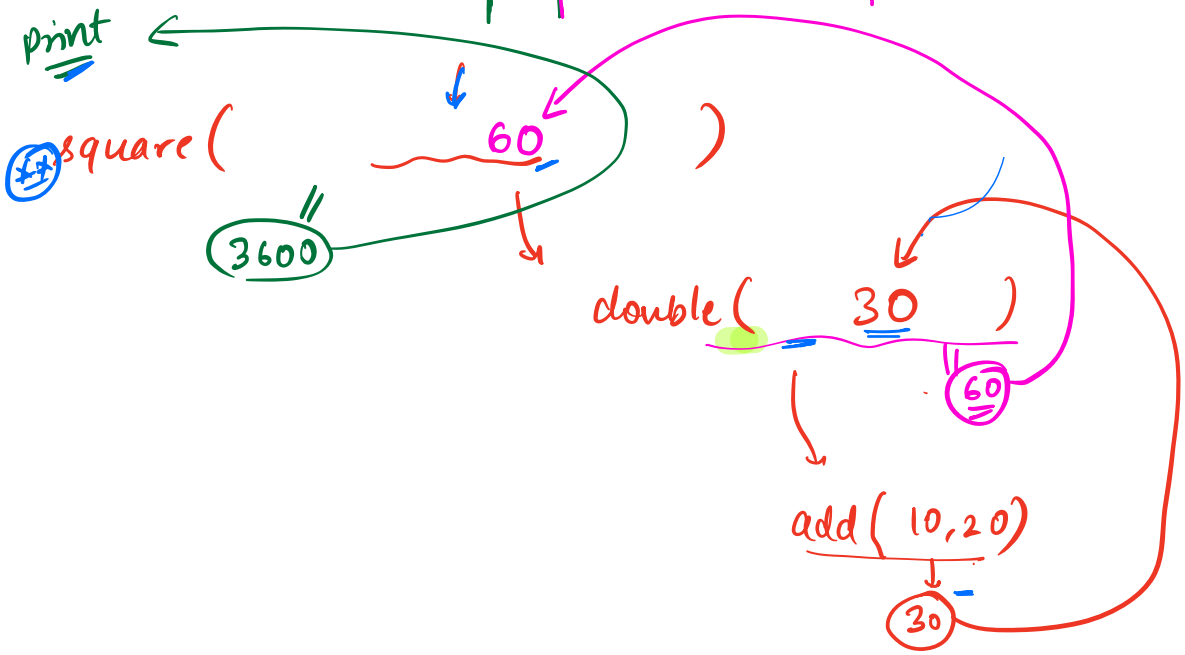
```
def add(a,b):
    return a+b

def double(a):
    return a*2

def square(a):
    return a*a
```

```
def main():
    x = add(10,20)
    y = double(x)
    z = square(y)
    print(z)
```

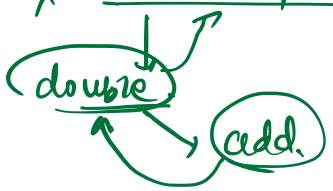
* first add → double → square. → print

Function calls are happen sequentially

```
def main():
    print(square(double(add(10,20))))
```



* Square gets executed fist but is not completed.

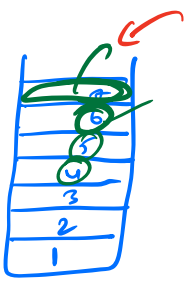


f1(f2(f3(f4(f5(f6(f7(inp)))))))

* Data structure to save function calls:

→ stack

Stack



Only one opening.

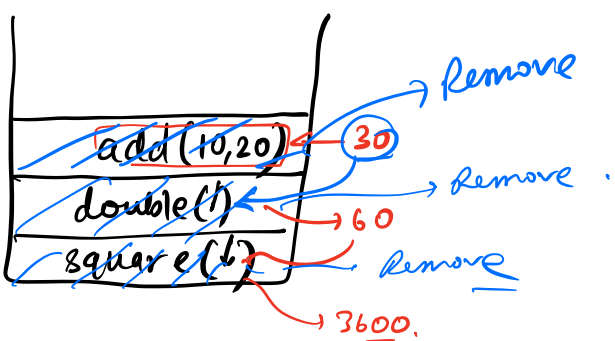
Last in, first out

LIFO

Push.

Pop

Stack

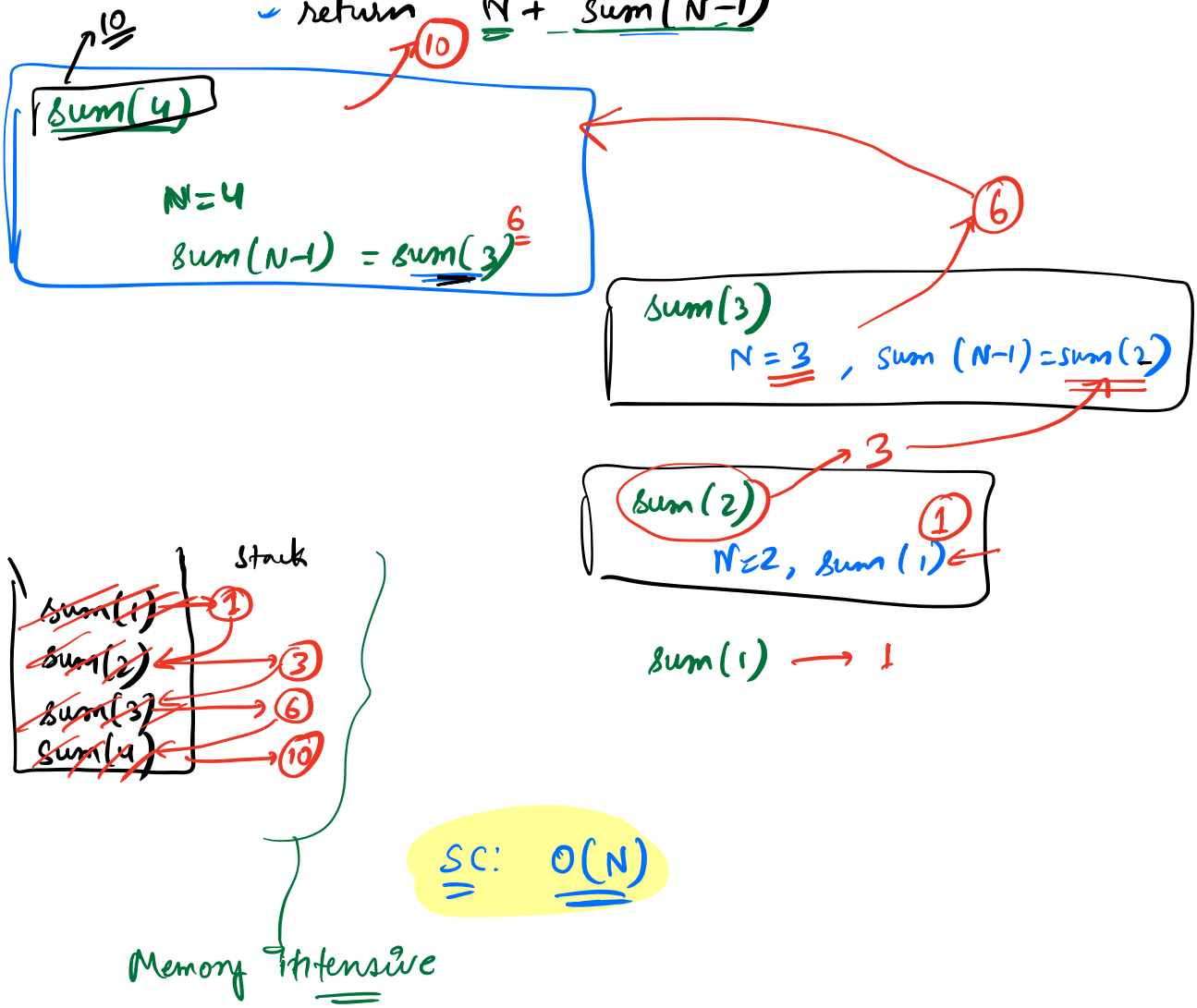


sum(N):

sum of 1+2+3...N

How does recursion work ?

```
int sum(N):
    if N == 1:
        return 1
    return N + sum(N-1)
```



SC: O(N)

Memory intensive

* what will happen if there is no base condition ?

→ <Stack overflow> ←

Break till 10:15

Q. Given N, print all nos. from 1 to N.

** Assumption:

* def printInc(N):

// print all nos. from 1 to N.

** Main Logic:

** { Print all nos. from 1 to N-1
  { Print Nth no.

1 2 3 4 5 6 ... N

** { printInc(N-1)
  { print(N)

** Base condition:

```
if N==1:
    print(1)
```

Code

```
def printInc(N):

    if N==1:
        print(1)
        return
    printInc(N-1)
    print(N)
```

N=1 printInc(0) X

1 2 3 4 5 ... N-1 N

H.W. Try seeing the output by building up the stack.

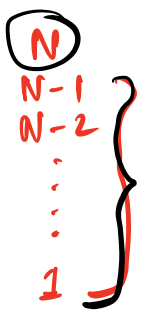
Q. Print all nos. from N to 1.
N, N-1, N-2, ... 3, 2, 1

Assumption

def printDec(N):
↓
N, N-1, N-2 1

Main Logic.

print(N)
printDec(N-1)



Code

```
def printDec(N):
    if N==1
        print(1)
        return
    print(N)
    printDec(N-1)
```

Dry Run is a little difficult in recursive codes.

N, N-1, ... 3, 2, 1

print(N)
printDec(N-1)

Q. Given a string, check whether it is palindrome or not.

→ a b c b a

2 pointer approach.

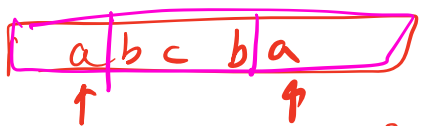
Recursive approach:

Assumption:

def isPalindrome (string, start, end)

Main logic:

```
if st[0] == st[-1] :
    isPalin( st[1:-1] )   <- // Extra space.
```



start = 1
end = 3

bcb

@ b c b @

```
if string[start] == string[end] :
    return isPalindrome(string,
                        start+1,
                        end-1)
else
    return False
```

Base condition

start == end ✓

```
if start >= end :
    return True.
```

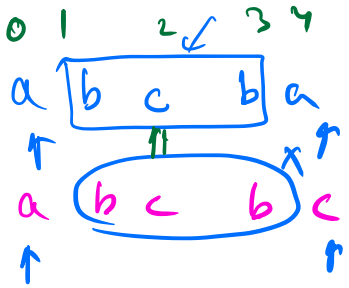
Code :

```
def isPalin( start, end, string)

    if start >= end :
        return True

    return string[start] == string[end]
           and isPalin( start+1, end-1, string)
```

```
def checkPalindrome ( string) :
    isPalin( 0, len(string) - 1, string)
```

start = 0, end = len(string) - 1

1 , len(- 2



("abcba", 2, 2)

Q. Given a directory structure, search for a file in it.

Google.

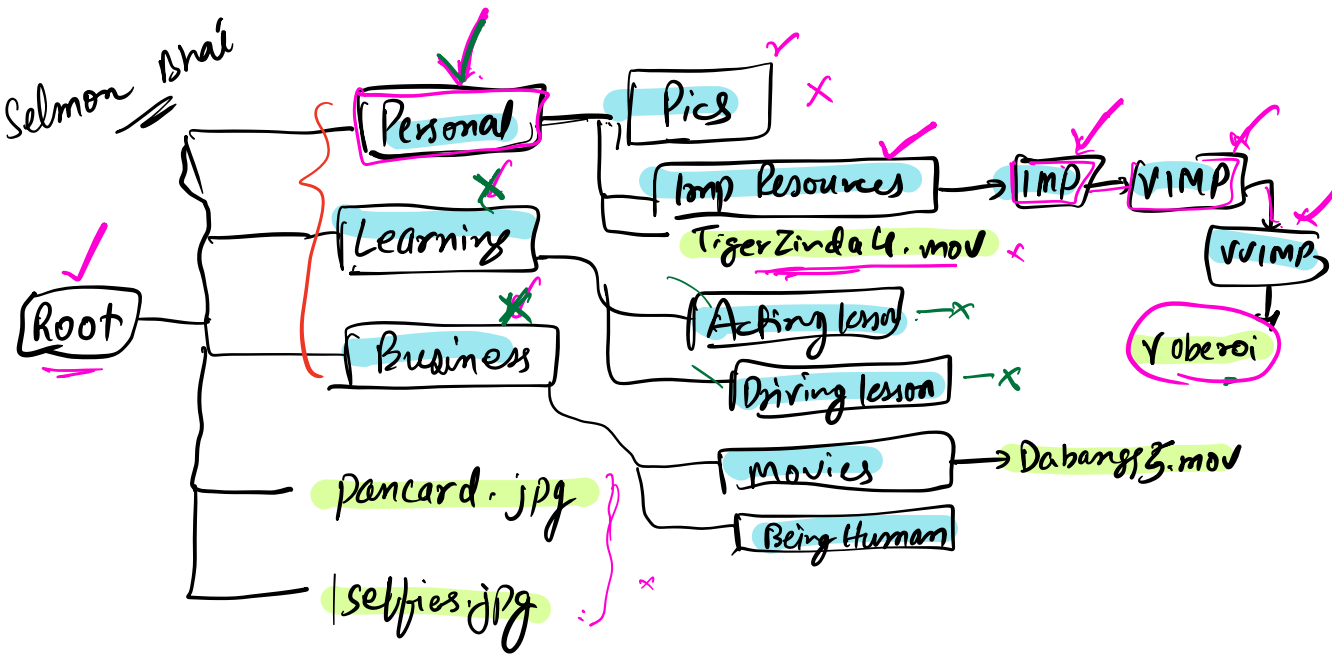
Inbuilt functions:

1. GetAllDirectories (DirectoryName)

↳ Returns all immediate directory names

GetAllFiles (DirectoryName)

↳ Returns all immediate files in this directory.



* GetAllDirectories (Root) → Personal, Learning, Business.

* GetAllFiles (Personal) → TigerZinda4.mov

SearchFile(root, TargetFile)

↘ return whether a file is present or not.

SearchFile(root, 'Aish') → No

SearchFile(root, "TigerZinda4") → Yes.

// Assumption:

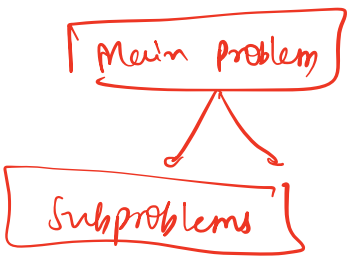
SearchFile(DirectoryName, TargetFile)

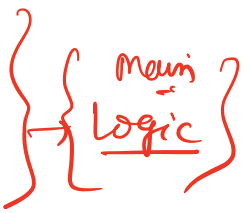
↓ ↘ return True/False.

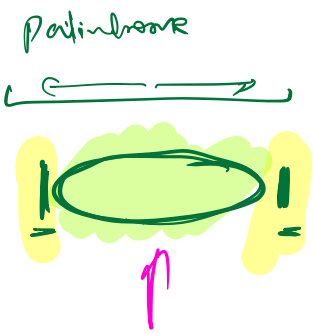
```
bool search(DirectoryName, TargetFile):
    dir_files = GetAllFiles(DirectoryName)
    if TargetFile in dir_files:
        return True

    directories = GetAllDirectories(DirectoryName)

    for dir in directories:
        res = search(dir, TargetFile)
        if res == True:
            return True
    return False.
```

Personal Business learnings.

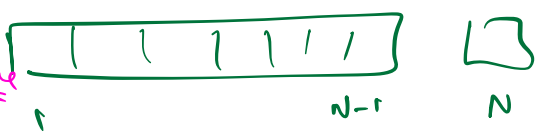
O(N)







Sum of A + B.



xx

"hello"

h - ello

ello

hello

hello
hell
hel
he
h.