RECURSION

(Function calling Itself)

- How do function calls work?

```
public class Main {
    public static void main(String[]args) {
        print_num(1);
    }
    static void print_num(int n) {
        System.out.println(n);
        print_num2(2);
    }
    static void print_num_2(int n) {
        System.out.println(n)
        print_num_3(3);
    }
    static void print_num_3(int n) {
        System.out.println(n)
    }
}
```

output:

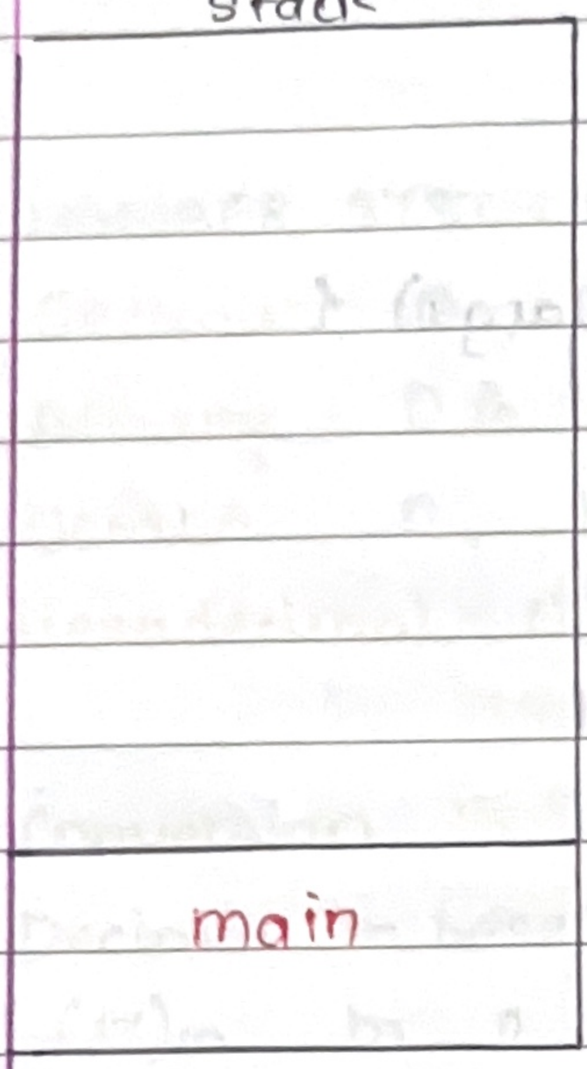
1

2

3

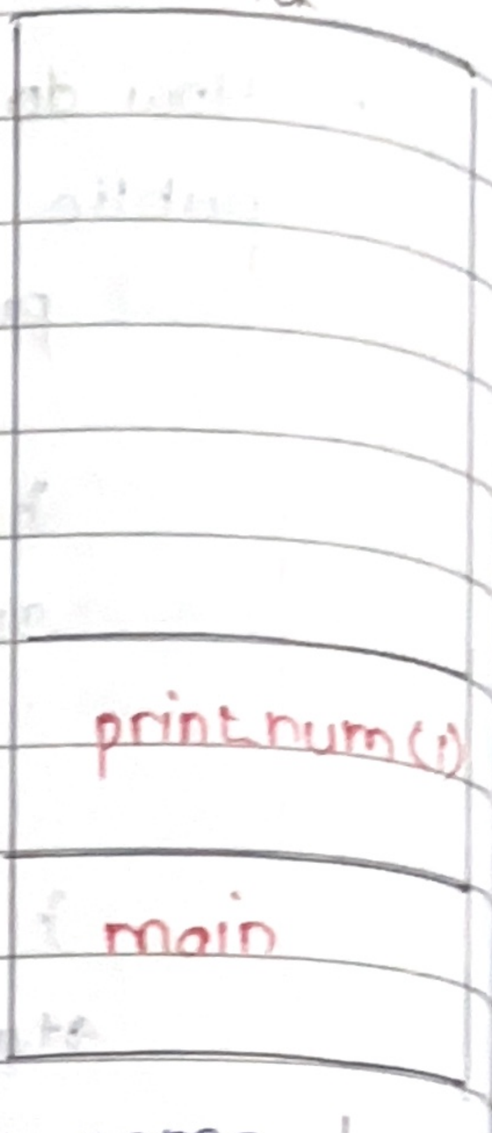
(1) While the function is not finished executing it will remain in the stack memory.

Stack



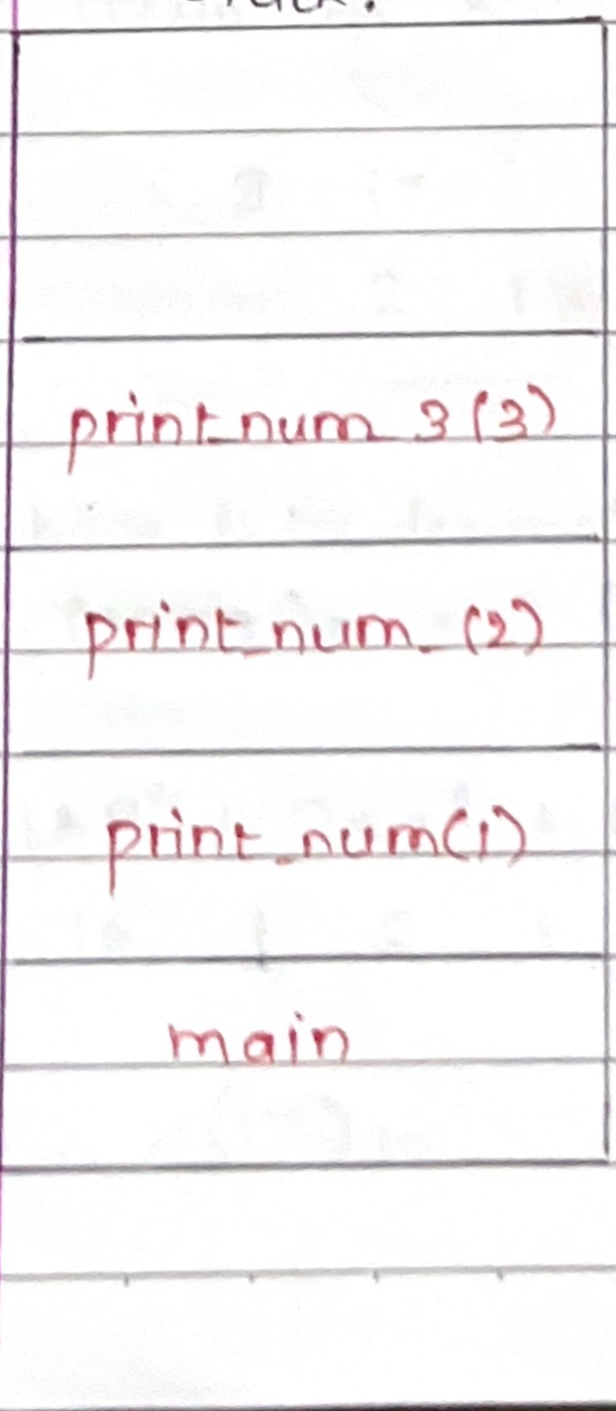
- main function will be the first function to be stored in the stack memory
- In our case it calls another function named print_num(1)
- After calling the function, it will not be removed from stack, instead it will wait until print_num has finished execution.

print_num(1)
Stack

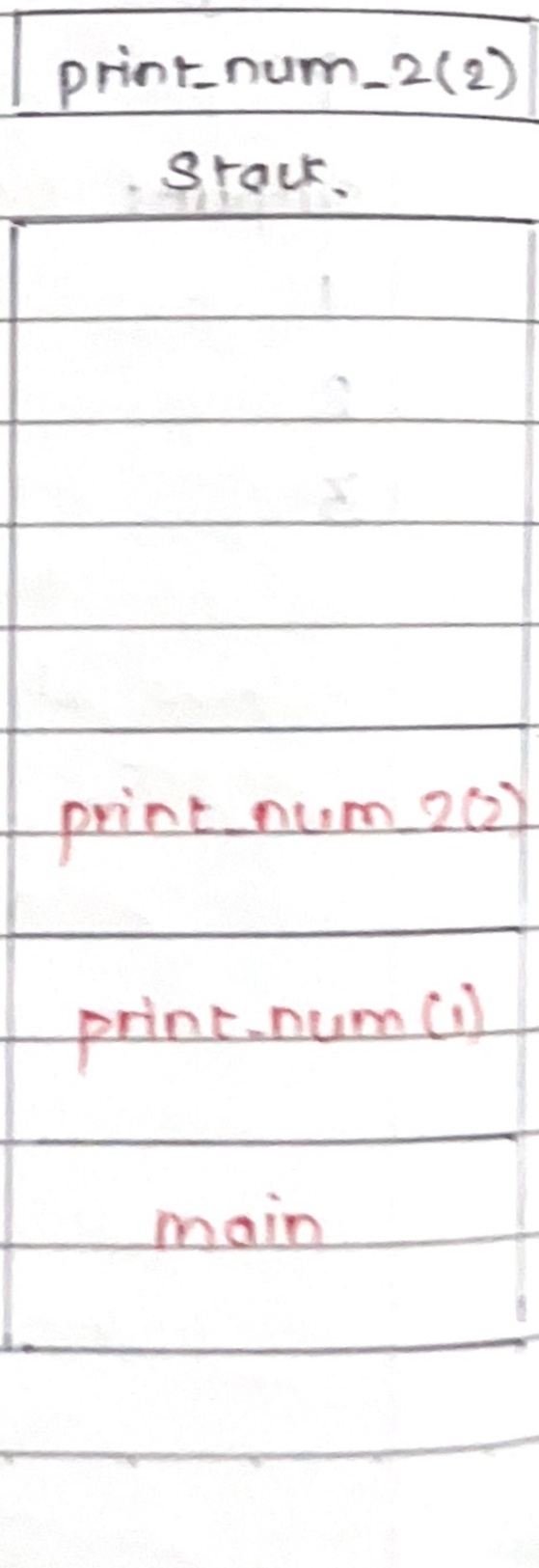


- print_num(1) will first print 1 and will do the same thing as the main function. It will tell print_num_2(2) function to finish its work and until then it will just wait in the stack memory

print_num_3(3)
Stack.

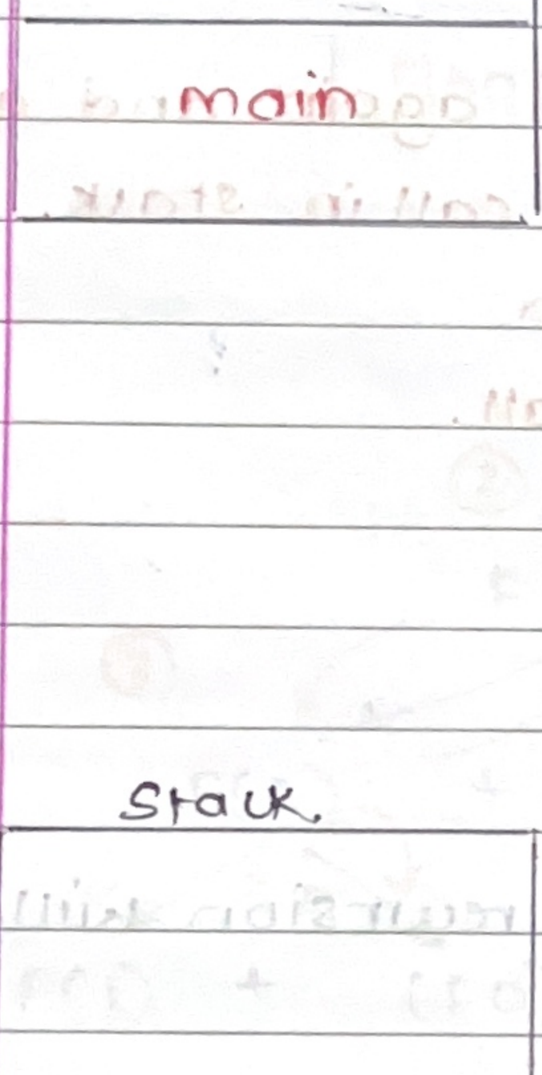


print_num_2(2)
Stack.

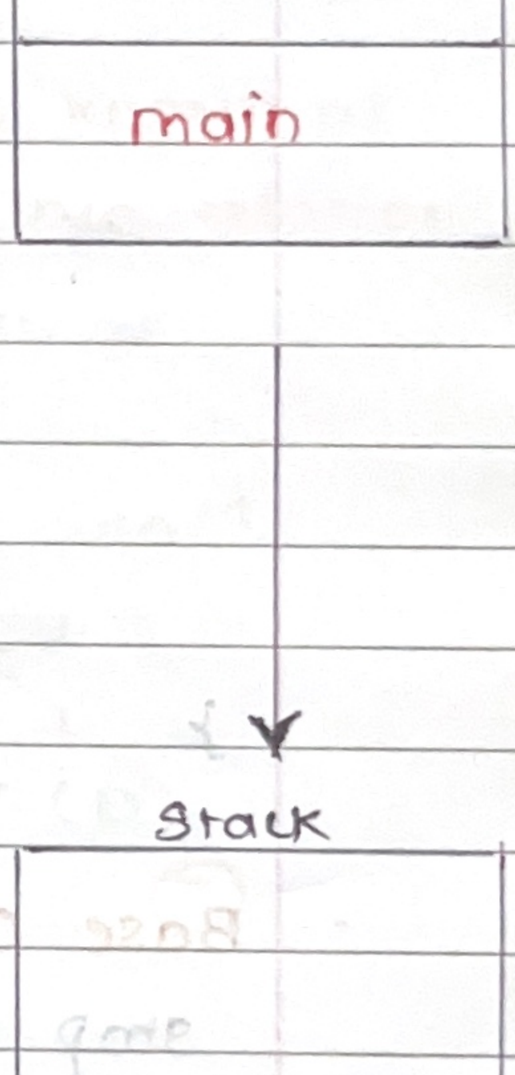


(2) When the function finishes execution it is removed from stack and the program flow is restored to where that function was called previously.

Stack.



Stack



Stack.

Stack

- Recursion function for the same.

```
public class NumberExampleRecursion {
    public static void main (String[]args) {
        print(1);
    }
    static void print(int n) {
        if (n==3) {
            System.out.println(3);
            return;
        }
        System.out.println(n);
        //recursive call
        // if you are calling a function again and again
         you can treat it as seprate call in stack.
        // this is called tail recursion.
        // this is the last function call.
        print(n+1);
    }
}
```

- Base condition - condition where our recursion will stop new calls

No Base condition - function calls will keep happening stack will be filled again and again
Memory of computer will exceed the limit - StackOverFlow error

M T W T F S S

Page No.:

Date:

YOUVA

Why Recursion?

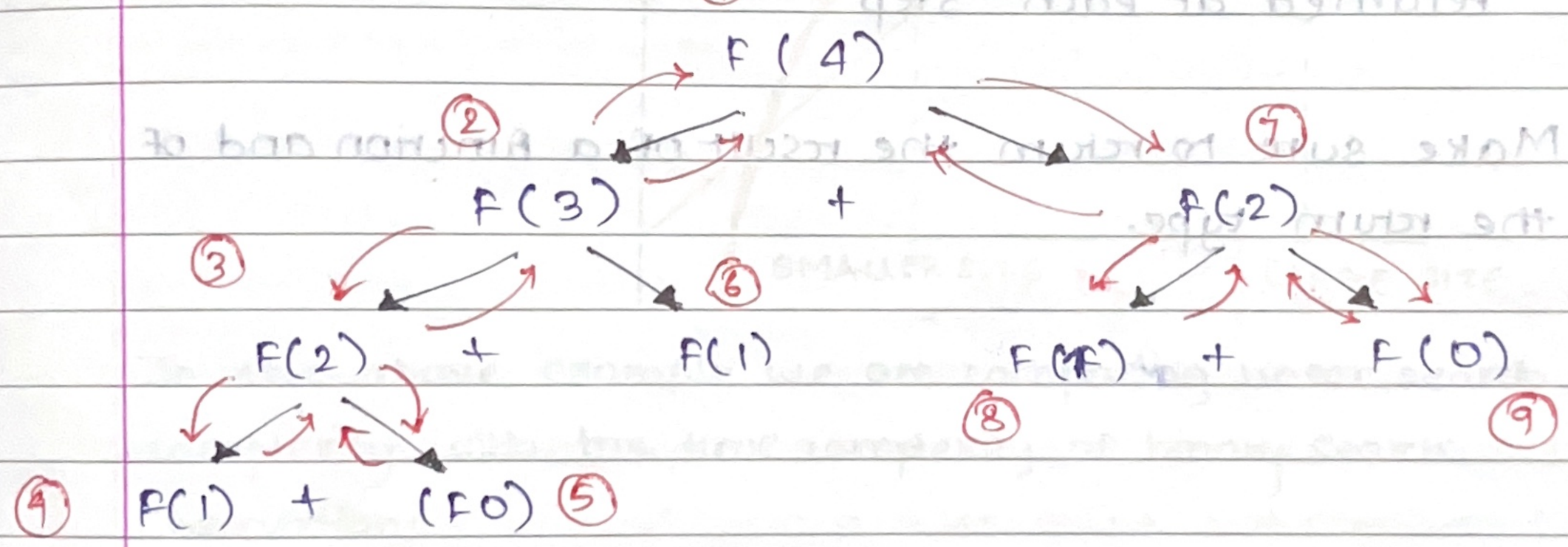
- It helps in solving bigger / complex problems in simple way
- You can convert recursion solution into iteration and vice versa.
- Space complexity is not constant because of recursive calls.
- It helps in breaking down bigger problems into smaller problems

Find Nth Fibonacci number

0th 1st 2nd 3rd 4th 5th 6th 7th

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

Fibo(N) = F(N-1) + Fibo(N-2) → This is known as recurrence relation



* The base condition is represented by answers we already have

In this case we know that F(0) = 0 , F(1) = 1

Note - How to understand and approach the problem

1. Identify if you can break down problem with into smaller problem
2. Write the recurrence relation if needed
3. Draw the recursion tree
4. About the tree.
   - See the flow of the functions, how they are getting in state
   - Identify and follow on left tree calls and right tree calls.
   - Draw the tree and pointer again and again using pen and papper
   - use a debugger to see the see the flow
5. see how the value and what type of value are returned at each step

Make sure to return the result of a function and of the return type.