Introduction to Recursion

* Let's understand Recursion with an example -

Q. Write a function that takes in a number and prints it.
// Print first 5 numbers ie. 1, 2, 3, 4, 5.

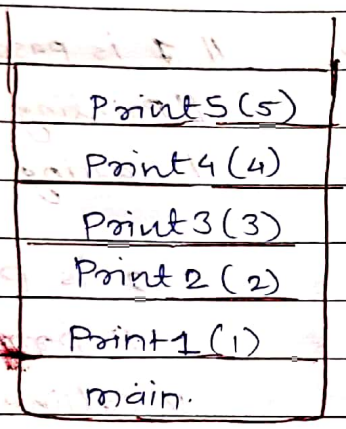
```
Public class example {
    Public static void main (Strings [] args) {
        Print1 (1);                    // it will call print1();
    }
    static void Print1 (int n) {       // 1 is passed here.
        System.Out.Println (n);        // Printing 1;
        Print2 (2);                    // calling print2();
    }
    static void Print2(int n) {        // 2 is passed here.
        System. out. Println (n);      // printing (2).
        Print3 (3);                    // calling print3();
    }
    Static void Print3 (int n) {       // 3 is passed here
        System. out. Println (n);      // printing 3
        Print4 (4);                    // calling print4();
    }
    Static void Print4 (int n) {       // 4 is passed here.
        System. out. Println (n);      // printing 4
        Print5 (5);                    // calling print5()
    }
    Static void Print5 (int n);        // 5 is passed
        System. out. Println. (n);     // printing 5.
    }
}
```

Lets see what happening in the above example:-

From back.

* Here one function calling another function.
* All this functions have same body & defination i.e taking one parameter & doing same thing.

* Working of a Function call:-
All the function calls that happen in a programming language, they go into the Stack Memory.



Stack

Output
1
2
3
4
5

* NOTE *. While the function is not finished execution, it will remain in the stack.

* When the function finishes executing it is removed from the stack & the flow of program is restored to where the function was called.

* if all the functions in previous example have same body & doing same things then why are we creating again & again.

* Solution:- call the function itself.

* What is Recursion ?

→ Function calling itself.

* Recursive Function for previous example:-

```
Public class example {
    Public static void main (string [] args) {
        Print (1);
    }
    static void Print (int n) {
        System.out.println (n);
        Print (n+1);          // Recursive call.
    }
}
```

→ But this function call will never stop, it will keep going. we are not stopping it anywhere. so in order to do that we need a **Base condition.**

* **Base Condition in Recursion:-**
A Condition where our recursion stop making new calls.

* **Solution:-**

```
Public class example {
    Public static void main (string [] args){
        Print(1);
    }
    static void Print (int n){
        if (n==5){                    // Base Condition.
            System.out.println(5);
            return;
        }
        System.out.Println (n);
        Print(n+1);                   // Recursive call
    }
}
```

classmate
Date
Page

Note:- If we are calling a function again & again, we can treat it as a separate call in a stack. Means, as many time we call the function it will take memory separately.

* No Base Condition →
Function calls will keep happening, stack will be keep getting ~~field~~ filled, And we knows every call of function will take some memory one time will come when memory of Computer will exceed the limit. This will give "Stackoverflow ERROR."

* Why Recursion ?

⟹ It helps us in Solving bigger / complex problems in a simpler way.

⟹ We can Convert recursion solution into iteration (loops) and vice-versa (because directly Solving the problems into ~~in-ter~~ iteration is difficult)

⟹ Space Complexity is not Constant because of recursive calls.

* Visualising Recursion :- (Previous example)

Program start main ()

↓

Print (1)

↓

Print (2)

↓

Print (3)

↓

Print (4)

↓

Print (5)

// Recursion Tree.

* How to understand and approach a problem:-

1) Identify if you can break down the problem into smaller one.

2) Write the recurrence relation if needed.

[Ex. Formula for Fibonacci numbers.

$F(n) = F(n-1) + F(n-2)$ ← This is recurrence relation]

3) Draw the recursive tree.

4) About the tree:

* See the flows of functions, how they are getting in stack.
* Identify & focus on left tree calls and right tree calls.
** Draw the tree & pointers again & again using pen & paper to understand problem.

** use a debugger to see the flows.

5) See hows the values are returned at each step see where the function call will come out. In the end, you will come out of the main function.

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

* Types of Recurrence Relation:-

(1) Linear Recurrance Relation :⟶ Ex. Fibonacci no.

(2) Divide & Conquer Recurrance Relation:-
⟶ Binary Search.