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 i.e, 1, 2, 3, 4, 5.

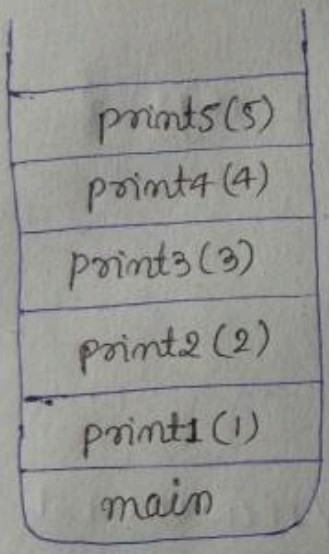
```
public class Example {
    public static void main(String[] args) {
        print1(1);                    // it will call print1()
    }
    static void print1(int n) {       // 1 is passed here
        System.out.println(n);        // it will print 1.
        print2(2);                    // it will call print2()
    }
    static void print2(int n) {       // 2 is passed here
        System.out.println(n);        // it will print 2.
        print3(3);                    // it will call print3()
    }
    static void print3(int n) {       // 3 is passed here.
        System.out.println(n);        // print 3
        print4(4);                    // call print4()
    }
    static void print4(int n) {       // 4 is passed here
        System.out.println(n);        // print 4
        print5(5);                    // call print5()
    }
    static void print5(int n) {       // 5 is passed here
        System.out.println(n);        // it will print 5.
    }
}
```

Let's see what is happening in above Example —

* Here, a function calling another function.
* All these functions have same body and definition i.e, taking one parameter and doing same things.

* Working of function calls :-

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 a function finishes executing, it is removed from the stack and the flow of program is restored to where the function was called.

NOTE : If all the functions in previous example have same body and 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.

* Generalised answer :-

```
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
    }
}
```

* NOTE :-
  - If we are calling a function again & again, we can treat it as a separate call in the 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 filled, And we know, 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 ?

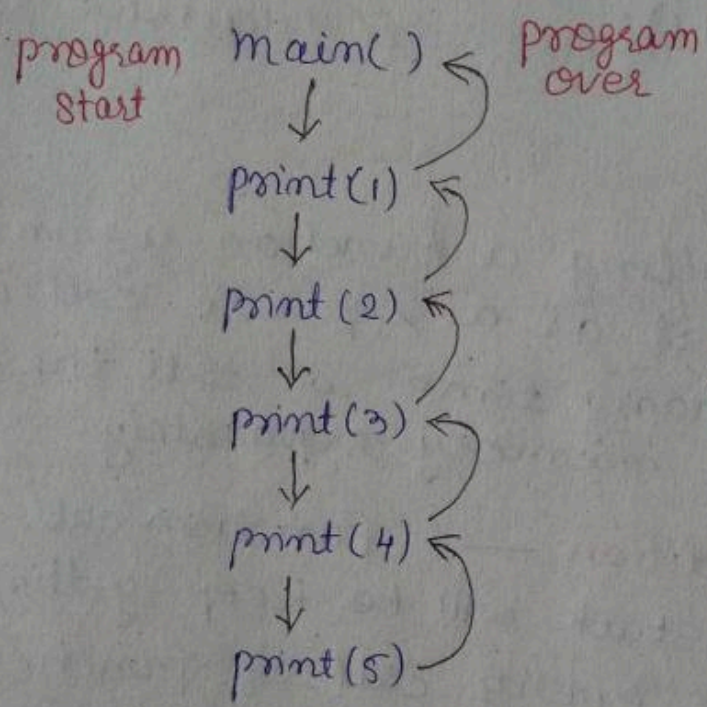
⇒ It helps us in solving bigger/complex problems in a simpler way.

⇒ We can convert recursion solution into ind Iteration (loops) and vice-versa (because directly solving the problems into interation is difficult).

⇒ Space complexity is not constant because of recursive calls.

* Visualising Recursion:

for previous Example



// Recursion Tree

* How to understand and approach a problem :—

1. Identify if you can break down the problem into smaller problems.
2. Write the recurrence relation if needed
   [Ex. Formula for Fibonacci numbers
   $F(n) = F(n-1) + F(n-2)$ ← This is the recurrence relation]
3. Draw the recursive tree.
4. About the tree :
   * See the flow of functions, how they are getting in stack.
   * Identify and focus on left tree calls and right tree calls.
   ** Draw the tree and pointers again & again using pen and paper to understand problem.
   ** use a debugger to see the flow.
5. See how 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 function and of the return type.

* Types of Recurrence Relation —

① Linear recurrence relation : ⟶ Example : fibonacci nos.

② Divide & Conquer recurrence relation :
Example → Binary search