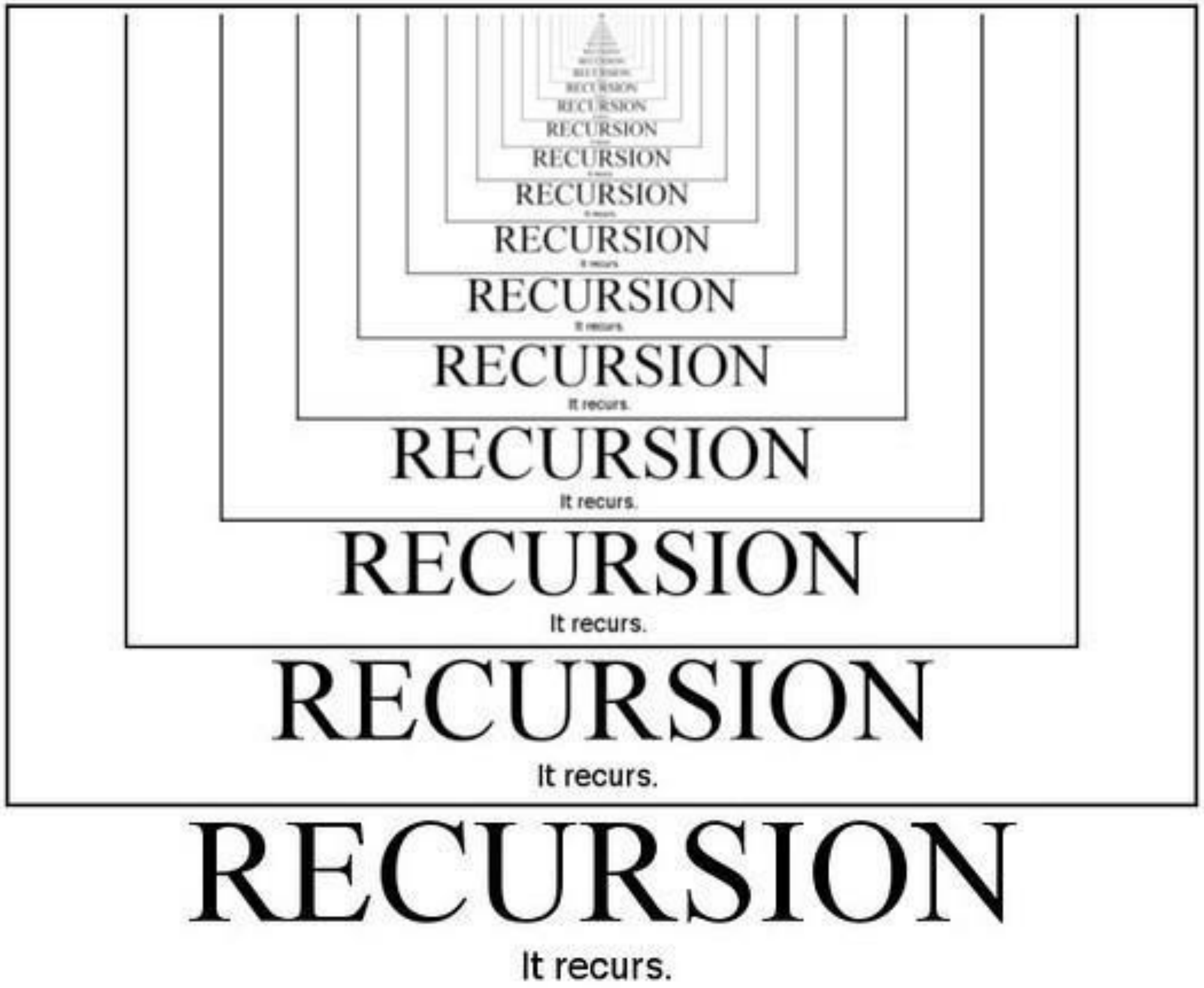
Data Structures and Algorithm


RECURSION
RECURSION
RECURSION
RECURSION
It recurs.
RECURSION
It recurs.
RECURSION
It recurs.
RECURSION
It recurs.
RECURSION
It recurs.
RECURSION
It recurs.
RECURSION
It recurs.
RECURSION
It recurs.

What is recursion?

- Recursion   is a process through which a function calls itself directly or indirectly again and again.

Why function calls itself?

- To break larger problem into smaller parts and solve it individually
- Reduces computational complexity
- Also called divide and conquer approach

- Recursive functions is defined in terms of base case and recursive steps

- **Base Case:** Simplest instance of a problem

- **Recursive steps:** We compute results by recursively calling a function and input size is decreased with each call.

Mathematical interpretation

- Problem: print sum of n natural numbers using recursion

F(n)=1+2+3+4+........................+(n-1)
+n

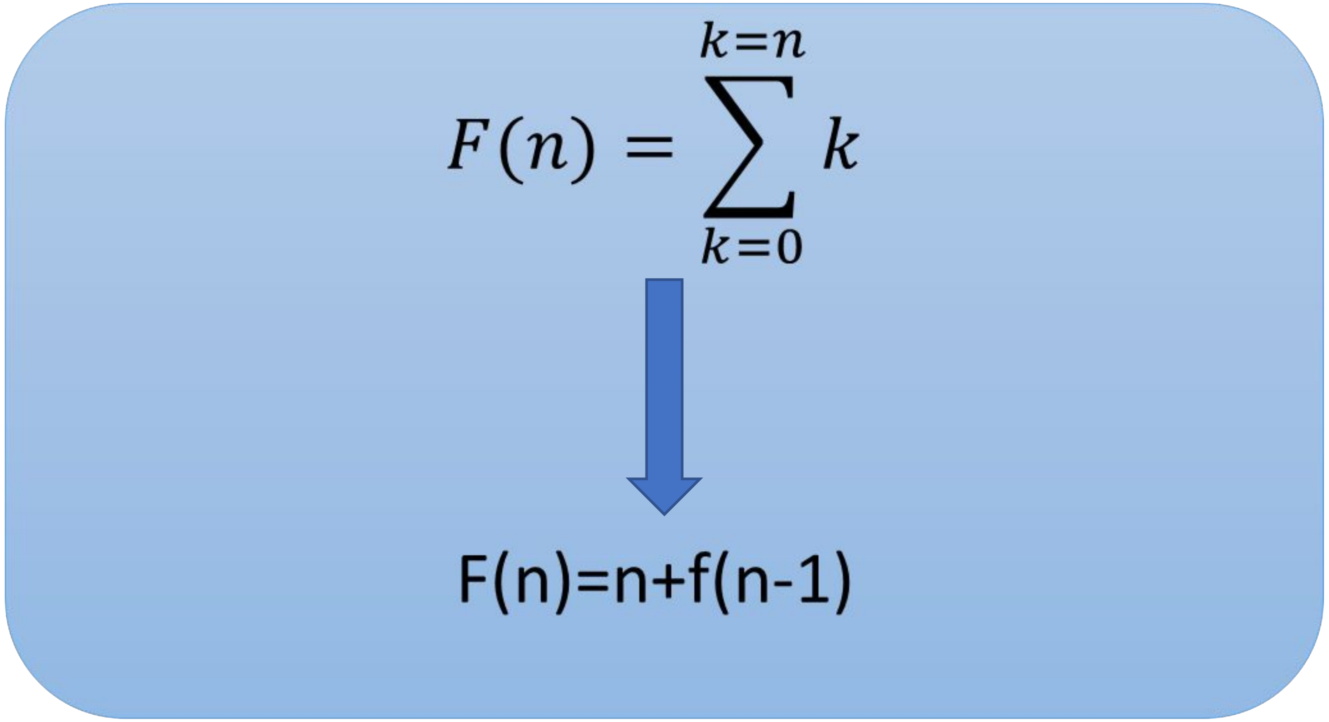
Mathematical interpretation

- Problem: print sum of n natural numbers using recursion

$$F(n)= \sum_{k=0}^{k=n} k$$

Mathematical interpretation

- Problem: print sum of n natural numbers using recursion



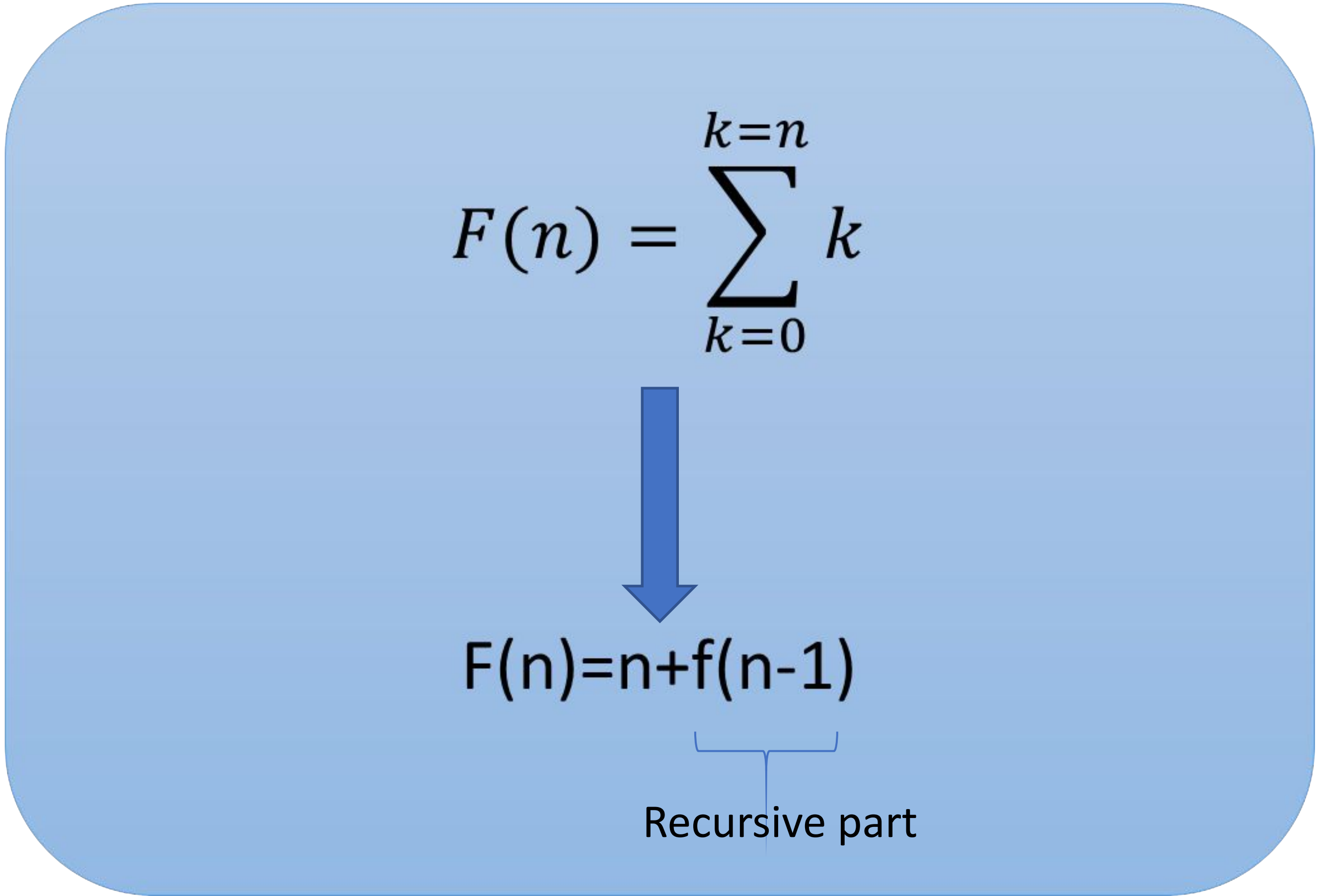
$$F(n) = \sum_{k=0}^{k=n} k$$

$$\downarrow$$

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

Mathematical interpretation

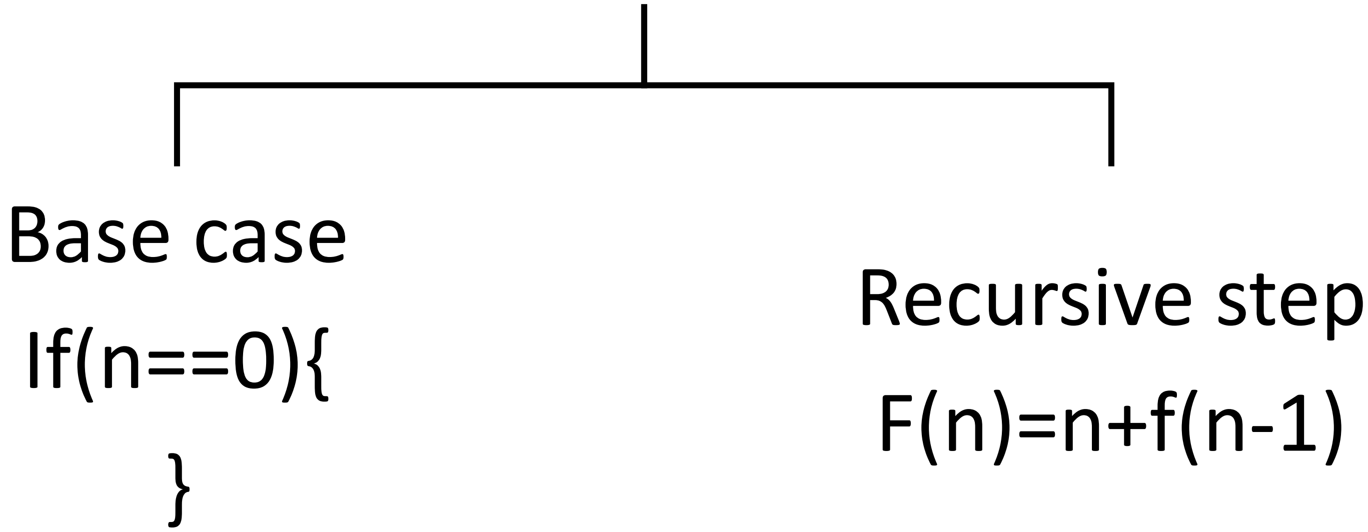
- Problem: print sum of n natural numbers using recursion



$$F(n) = \sum_{k=0}^{k=n} k$$

$$\big\downarrow$$

$$F(n)=n+\underbrace{f(n-1)}_{\text{Recursive part}}$$



Base case

If(n==0){

}

Recursive step

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

Recursion in memory

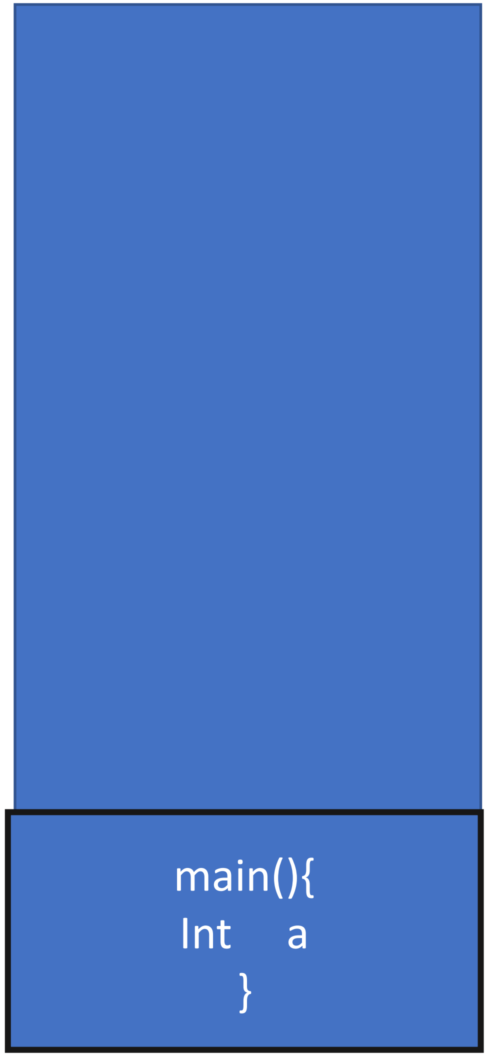
Recursive program in java

```
void func(int a){
     if(a==0){            //base case
    return;
    System.out.print(a);
      func(a-1);            //recursive step
      }
    }
Public static void main(string args[]){
 int a =3;
  func(a);
}
```

Recursive program in java

```
void func(int a){
     if(a==0){                    //base case
    return;
    System.out.print(a);
       func(a-1);                  //recursive step
       }
    }
Public static void main(string args[]){
 int a =3;
   func(a);
}
```

Stack

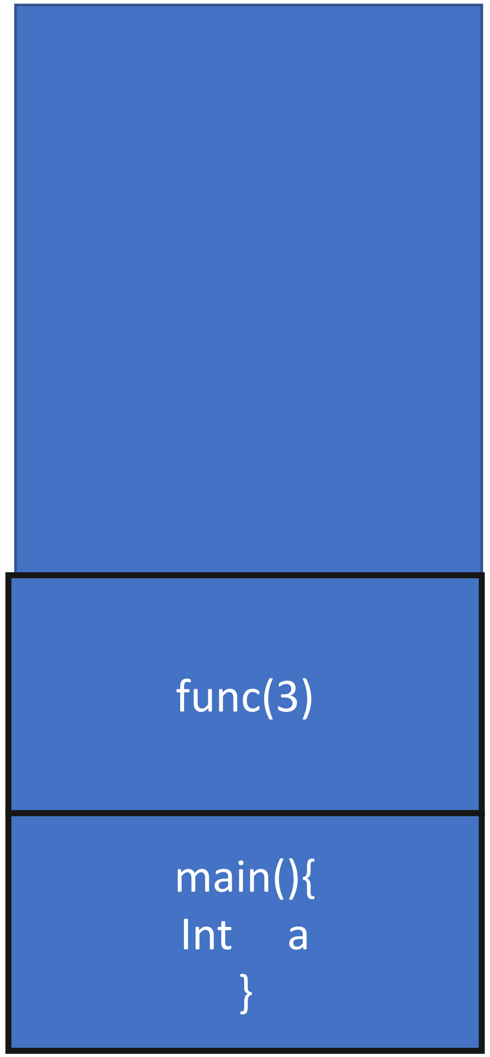


main(){
Int   a
}

Recursive program in java

```
void func(int a){
        if(a==0){                      //base case
      return;
      System.out.print(a);
          func(a-1);                   //recursive step
         }
      }
Public static void main(string args[]){
 int a =3;
   func(a);
}
```

Stack



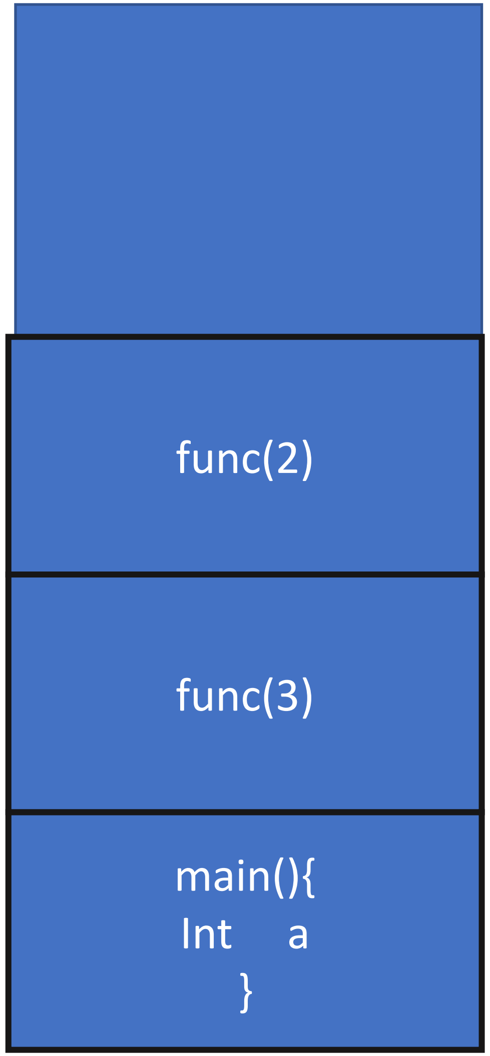
func(3)

main(){
Int   a
}

Recursive program in java

```
void func(int a){
         if(a==0){                          //base case
       return;
       System.out.print(a);
            func(a-1);                      //recursive step
           }
       }
Public static void main(string args[]){
 int a =3;
   func(a);
}
```

Stack



func(2)

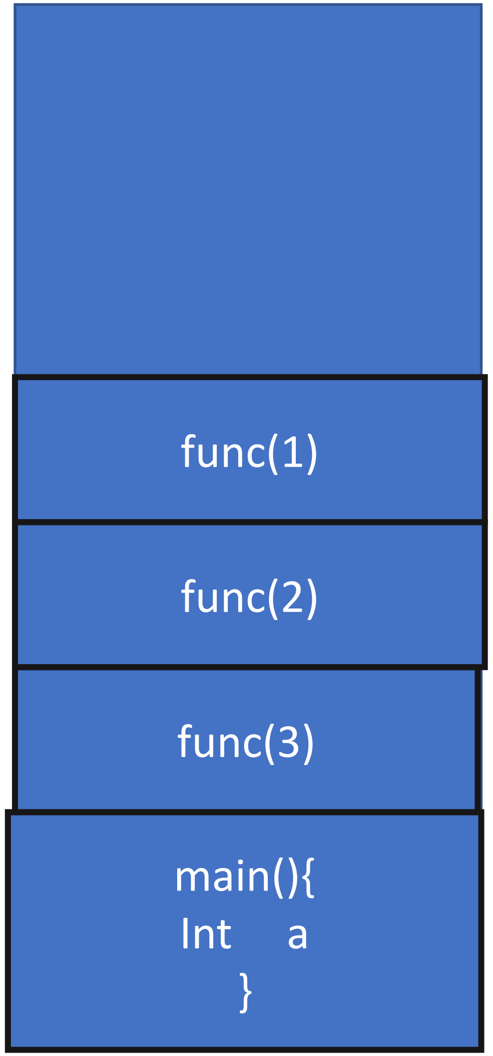
func(3)

main(){
Int   a
}

Recursive program in java

```
void func(int a){
         if(a==0){                      //base case
        return;
        System.out.print(a);
            func(a-1);                     //recursive step
           }
        }
Public static void main(string args[]){
 int a =3;
    func(a);
}
```

Stack



func(1)

func(2)

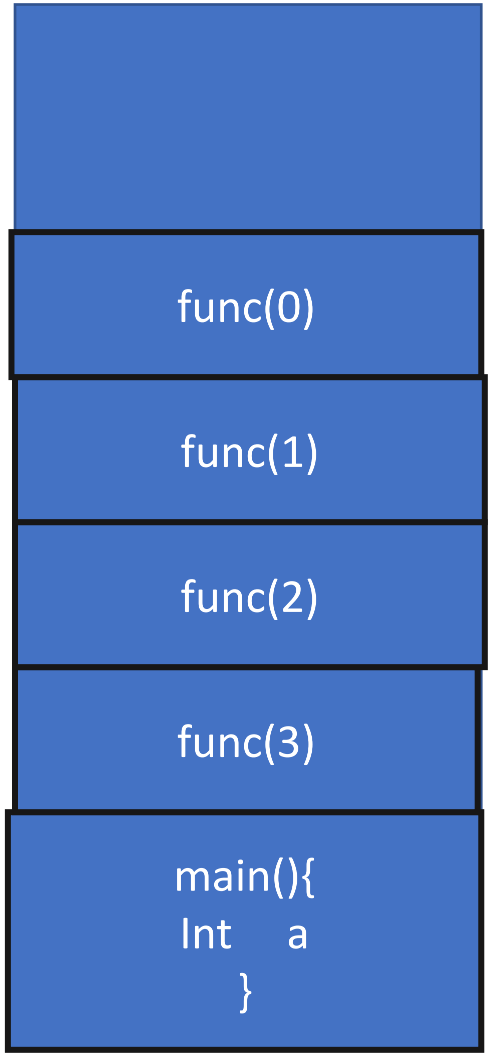
func(3)

main(){
Int a
}

Recursive program in java

```
void func(int a){
        if(a==0){                    //base case
       return;
       System.out.print(a);
           func(a-1);                    //recursive step
          }
       }
Public static void main(string args[]){
 int a =3;
   func(a);
}
```

Stack



func(0)

func(1)

func(2)

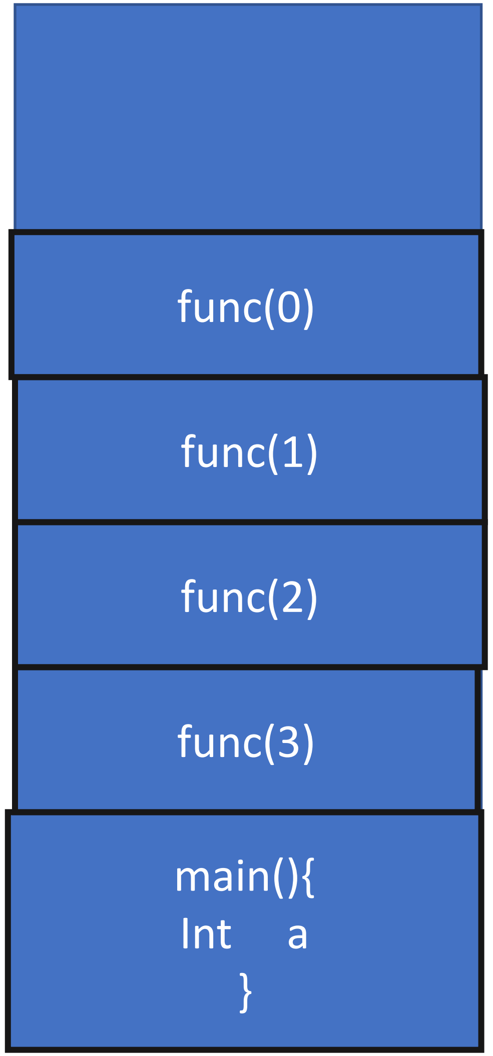
func(3)

main(){
Int   a
}

Recursive program in java

```
void func(int a){
        if(a==0){                    //base case
      return;
      System.out.print(a);
          func(a-1);                 //recursive step
         }
      }
Public static void main(string args[]){
 int a =3;
   func(a);
}
```

Stack



func(0)

func(1)

func(2)

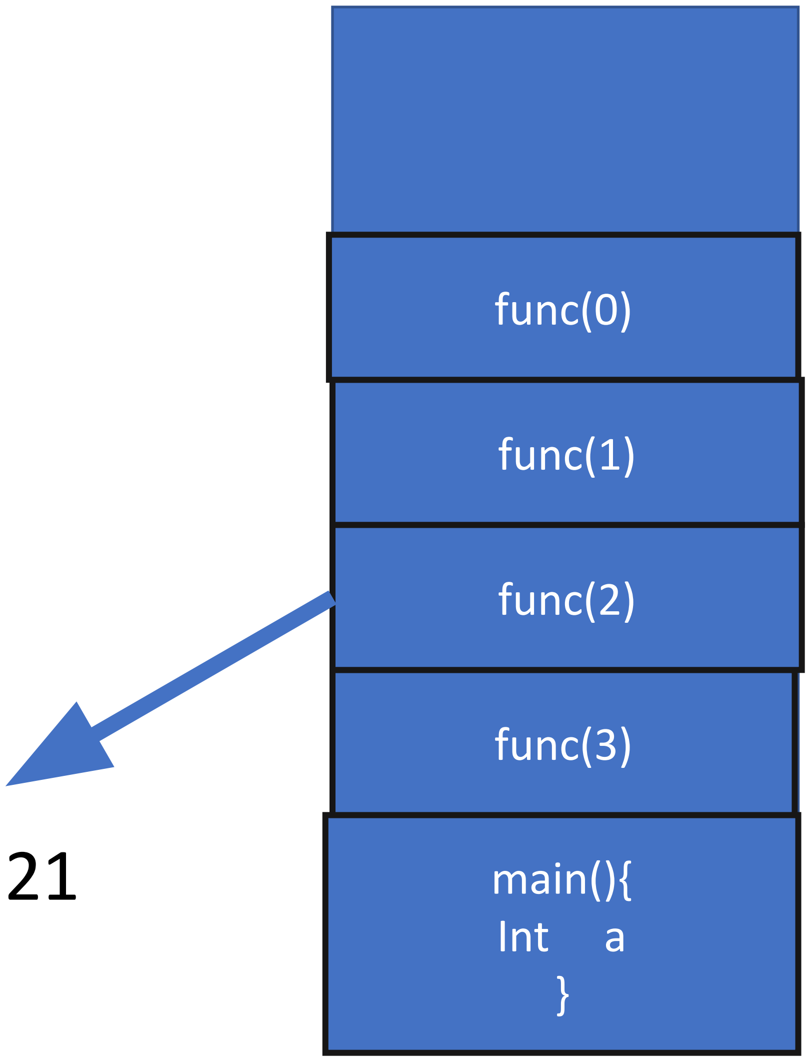
func(3)

main(){
Int a
}

Recursive program in java

```
void func(int a){
      if(a==0){                    //base case
     return;
     System.out.print(a);
        func(a-1);                   //recursive step
       }
     }
Public static void main(string args[]){
 int a =3;
   func(a);
}
```

Stack



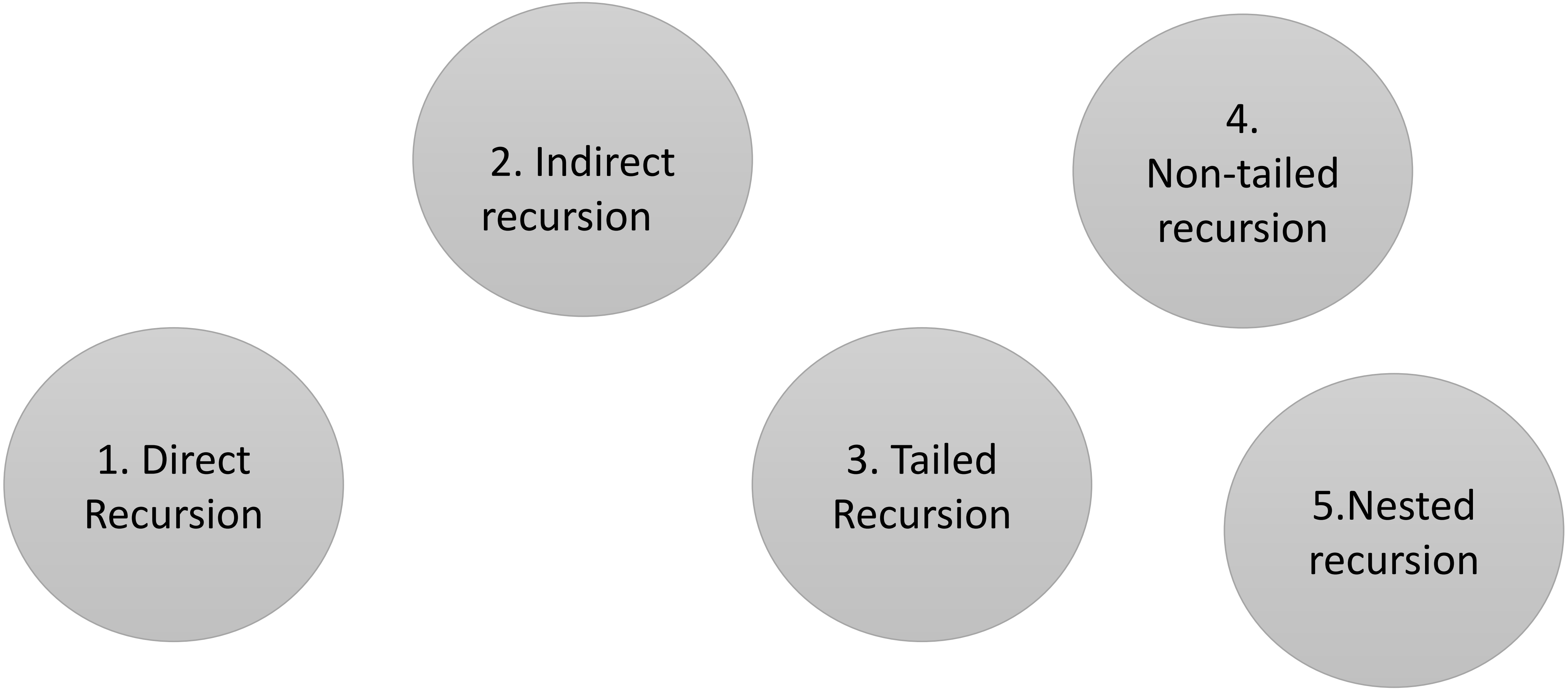
321

Think And Answer

- Based on properties of recursion, which of the following statement is right?

1. Recursion is always better than iteration
2. Recursion uses more memory compared to iteration
3. Recursion uses less memory compared to iteration.

Types of Recursion



2. Indirect recursion

4. Non-tailed recursion

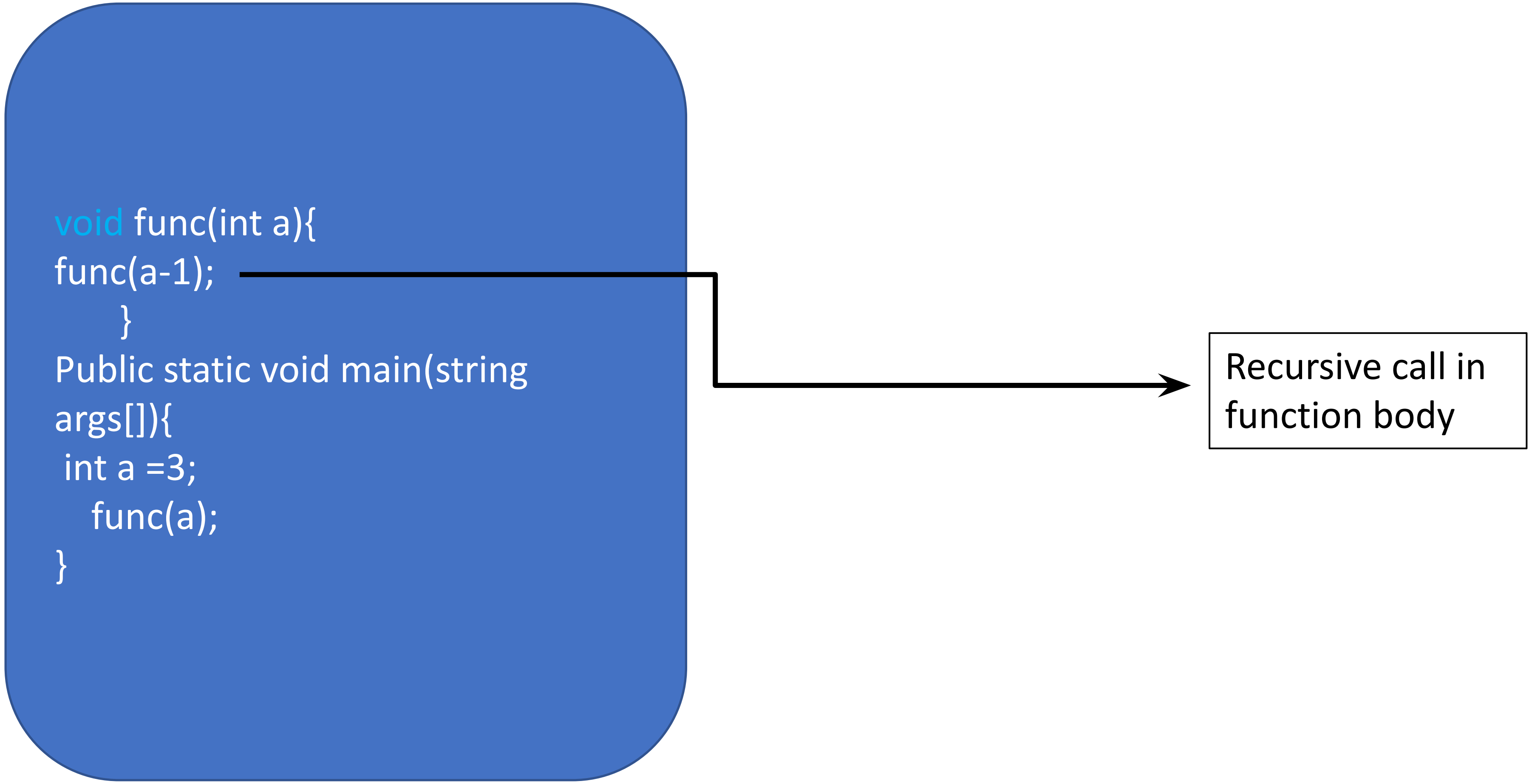
1. Direct Recursion

3. Tailed Recursion

5.Nested recursion

Direct Recursion

- A function is called direct recursive if it calls itself in its body again.

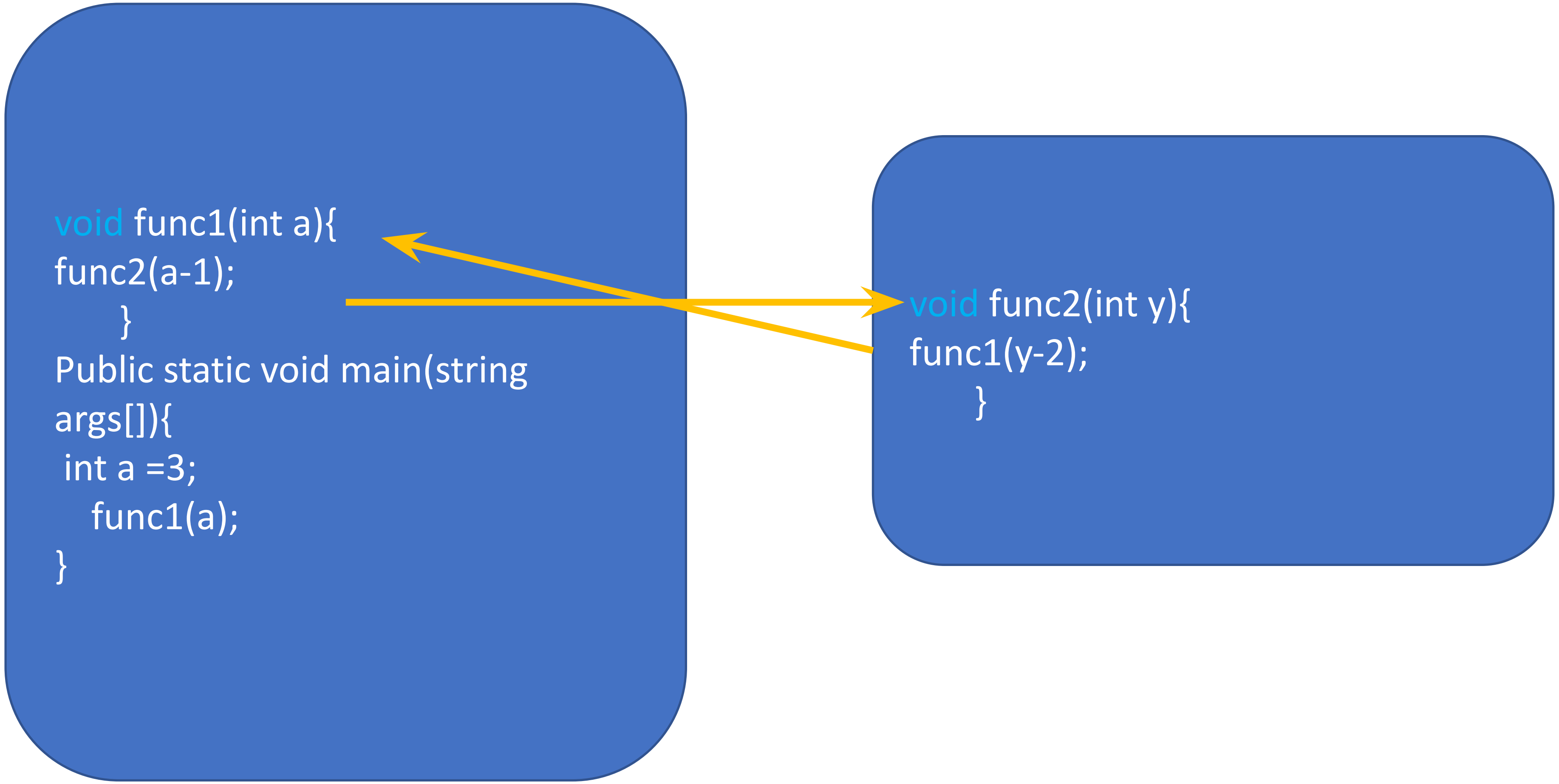


```
void func(int a){
func(a-1);
       }
Public static void main(string
args[]){
 int a =3;
    func(a);
}
```

Recursive call in function body

Indirect recursion

- The recursion in which a function calls itself via another function

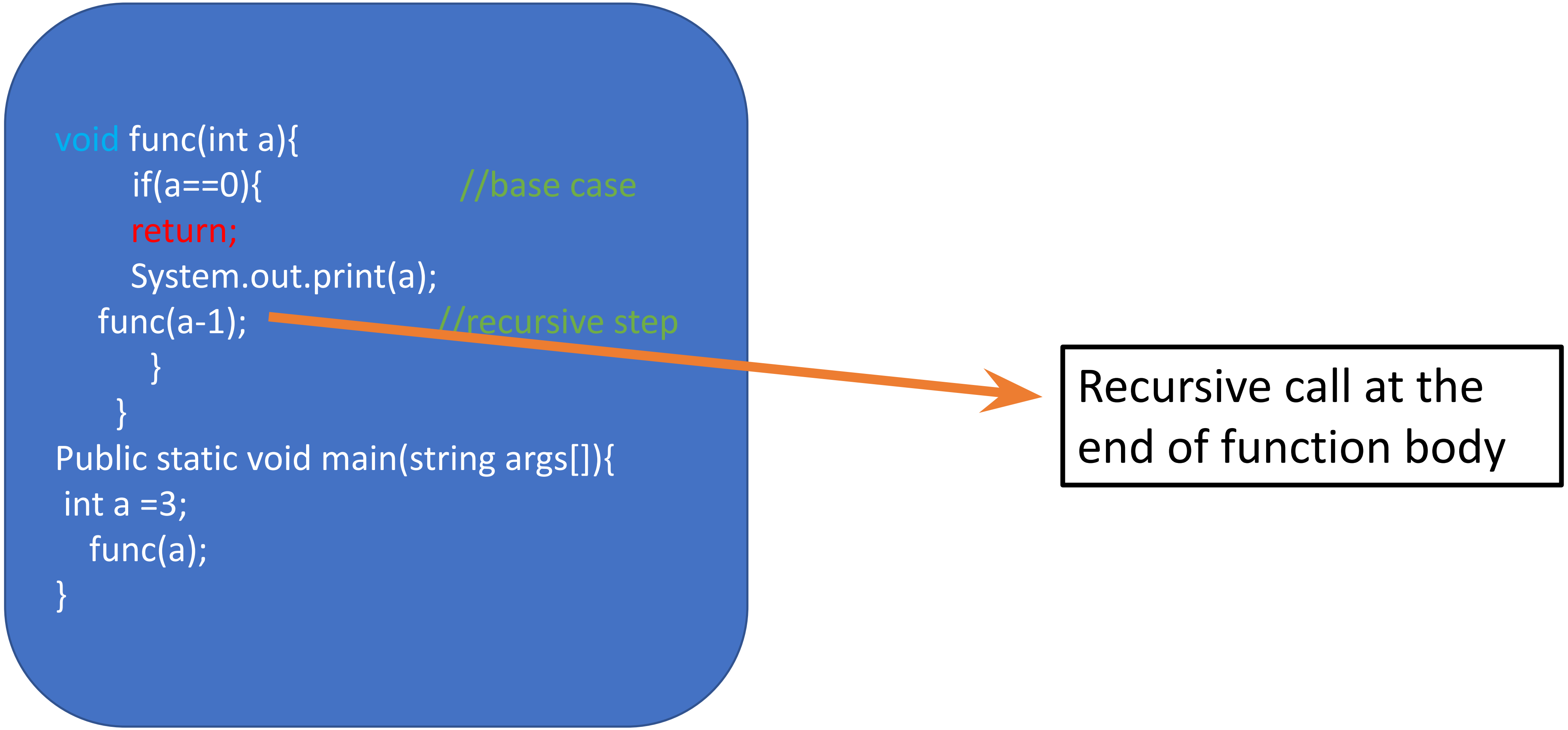


```
void func1(int a){
func2(a-1);
       }
Public static void main(string
args[]){
 int a =3;
    func1(a);
}
```

```
void func2(int y){
func1(y-2);
       }
```

Tailed Recursion

- A function which executes a recursive call at the end of its local body .



```
void func(int a){
        if(a==0){                  //base case
        return;
        System.out.print(a);
     func(a-1);                  //recursive step
          }
      }
Public static void main(string args[]){
 int a =3;
   func(a);
}
```

Recursive call at the end of function body

Non-Tailed Recursion

- If a function does not execute a recursive call   at the end is called non-tailed recursion

```
void func(int a){
func(a-1);                    //recursive step
System.out.print(a);

      }
Public static void main(string args[]){
 int a =3;
   func(a);
}
```

Nested Recursion

- Nested recursion, the recursive function will pass the parameter as a recursive call.

```
void func(int a){
If(){
func(func(a-1));
}
}
```

Example:

F(n)=1+2+3+4+........................+(n-1)
+n

Program to add sum on n natural numbers

```
public class Recursion {
    int temp=0;
   int func(int a){
       if(a==0){
           return 0;
       }
     temp=func(a-1);
     return a+temp;
   }
public static void main(String[] args){
Recursion r=new Recursion();
    System.out.print(r.func(5)+"\n");
} }
```

Advantages of Recursion

1. Usually code is simpler to write
2. Extremely useful when a task can be simplified into an easy action plus a simpler variant of the same task.
3. To solve problems which are naturally recursive

   for example - the tower of Hanoi.

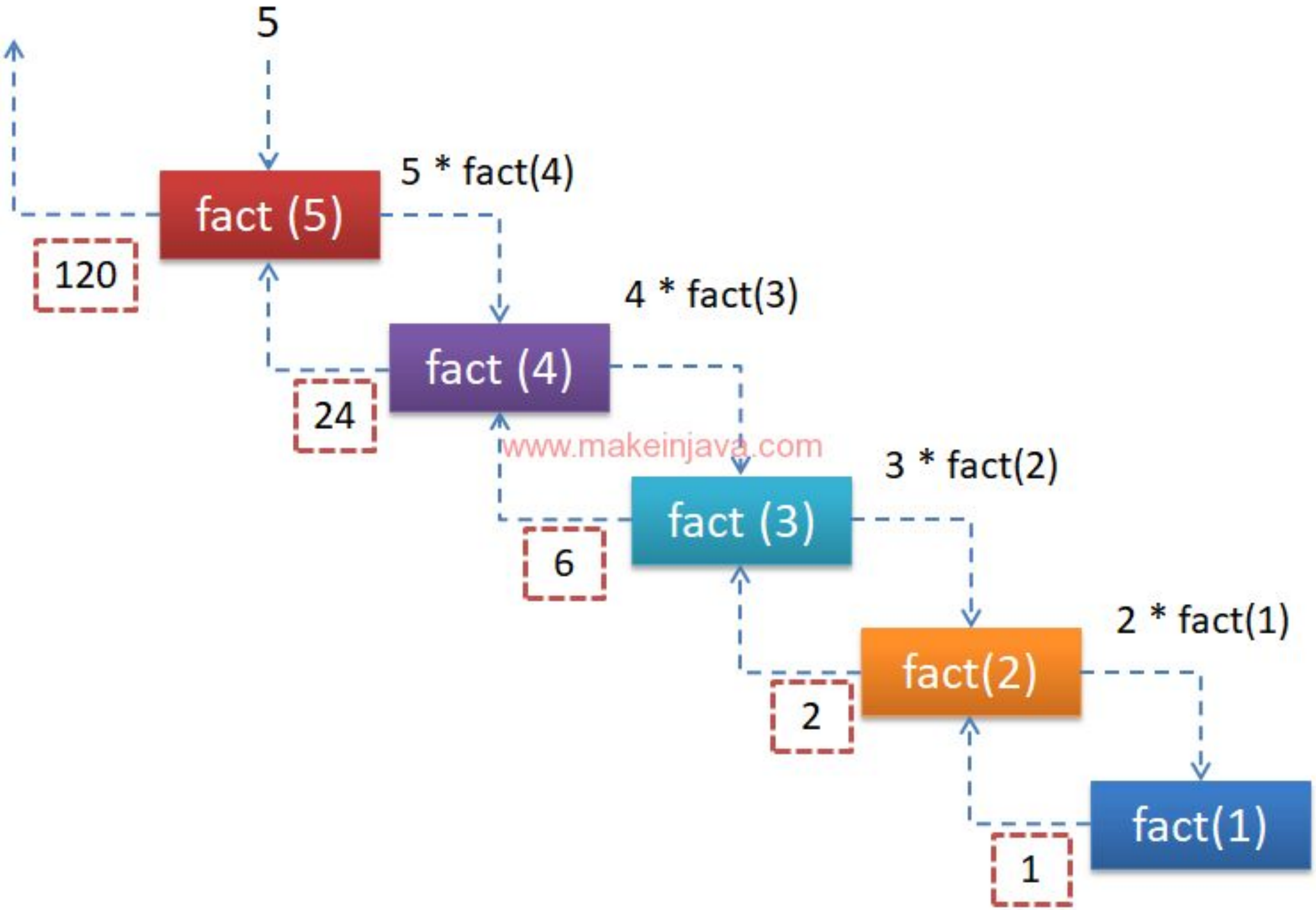
1. Recursion reduce the length of code.
2. Useful in solving data structures - tree related problems

Disadvantages of Recursion:

1. Recursive functions are inefficient in terms of space and time complexity
2. They require a lot of memory space to hold intermediate results on the system's stacks.
3. Recursion can sometimes be slower than iteration because in addition to the loop content, it has to deal with the recursive call stack frame. This will result in more code being run, which will make it slower.
4. The computer may run out of memory if the recursive calls are not properly checked(stack overflow), or the base case is not added.
5. Sometimes they are hard to analyze or it is difficult understand the code.

Applications of Recursion

- Tree Traversals
- Tree Problems: InOrder, PreOrder PostOrder
- Graph Traversals: DFS [Depth First Search] and BFS [Breadth First Search]
- Towers of Hanoi
- Backtracking Algorithms
- Divide and Conquer Algorithms
- Dynamic Programming Problems
- Merge Sort, Quick Sort
- Binary Search
- Fibonacci Series, Factorial, etc.



5
fact (5)
5 * fact(4)
120
fact (4)
4 * fact(3)
24
fact (3)
3 * fact(2)
6
fact(2)
2 * fact(1)
2
fact(1)
1