# Recursion level 1 Questions

Q1]. Print numbers from n to 1.

Soln:- If N=5. Print:- 5 4 3 2 1.

f(5) -> f(4) -> f(3) -> f(2) -> f(1).

Base condition:-

If n == 0
return.

Code:-

```java
public class Nto1 {
    public static void main(String[] args) {
        fun(n:5);
    }

    static void fun(int n) {
        if (n == 0) {
            return;
        }
        System.out.println(n);
        fun(n:n-1);
    }
}
```

Q). Print 1 to n.

Soln: If n = 5, Print 1, 2, 3, 4, 5.

Logic:-

f(5) -> f(4) -> f(3) -> f(2) -> f(1).

3rd 2nd Print first

Instead of calling recursive function after printing n, first call the recursive function and then print. So, that it gets printed from 1 to n.

Code.

```
static void funRev(int n) {
    if (n == 0) {
        return;
    }
    funRev( n: n-1);
    System.out.println(n);
}
}
```

Q1. To print n to 1 and 1 to n.

Soln:- Fore:- n = 5.

5 4 3 2 1 1 2 3 4 5.

Code:-

```
if (n == 0) {
    return
}
sout (n);
funRev (n-1);
sout (n);
```

---

Q). Factorial of a number:-

Soln: If n = 4,

Ans:- 4 x 3 x 2 x 1 = 24

Base condition:-

```
if (n == 1)
    return 1;
```

Recurrence Relation:- f(n) = n x f(n-1).

Logic:- N=4.

4x f(3)
↳ 3x f(2).
↳ 2x f(1)
(1).

Returning:-

f(2) = 2x1
f(3) = 3x2x1
f(4) = 4x3x2x1

---

Code:-

```
public class Fact {
    public static void main(String[] args) {
        int ans = fact(n: 5);
        System.out.println(ans);
    }

    static int fact(int n) {
        if (n <= 1) {
            return 1;
        }

        return n * fact(n: n-1);
    }
}
```

Q). Sum of n to 1

Soln:- For eg:- If n=4.

4+3+2+1 = 10

Base condition.

```
If (n==1)
    return 1;
```

Recurrence Relation:- n + f(n-1)

Code:-

```
public class Sum {
    public static void main(String[] args
        int ans = sum( n: 5);
        System.out.println(ans);
    }

    static int sum(int n) {
        if (n <= 1) {
            return 1;
        }

        return n + sum( n: n-1);
    }
}
```

Q. Sum of digits.

Soln:- Eg:- N = 1342.

Ans:- 1 + 3 + 4 + 2 = 10

Generic method:- First -> N % 10 -> To get last digit.

Loop till N = 0.
- Sum += first.
- N -> N/10 -> To remove last digit.

Recursion:- N = 1342

2 + F(134) -> N/10.

N % 10

F(134) -> 4 + F(13)

F(13) -> 3 + F(1).

Base condition:-

```
if (n == 0)
    return 0;
```

## Recurrence Relation:-

$$f(N) = f(N/10) + (N\%10)$$

- $f(N/10)$ → To pass remaining digits.
- $(N\%10)$ → To get last digit.

## Code:-

```java
public class DigitSum {
    public static void main(String[] args) {
        int ans = sum( n: 1342);
        System.out.println(ans);
    }

    static int sum(int n) {
        if (n == 0) {
            return 0;
        }
        return (n%10) + sum( n: n/10);
    }
}
```

Q). Product of digits:-

Soln:- N = 1342, Ans = 2 x 4 x 3 x 1
= 36

Logic:- 2 x F(134)
↓
N%10

F(134) → N/10.
↓
4 x F(13)
↓
3 x F(1).

Base condition:-

```
If(n==0)
    return 1;
```

How, when n==0 we do not return 0, as the final answer would become 0.

Alternate:-

Base condition:-

```
If(n%10==n)
    { return n;
    }
```

If one digit is remaining return that digit.

Code :-

```java
public class DigitProduct {
    public static void main(String[] args) {
        int ans = prod(n: 1342);
        System.out.println(ans);
    }

    static int prod(int n) {
        if (n%10 == n) {
            return n;
        }
        return (n%10) * prod(n: n/10);
    }
}
```

## Concept of passing numbers

Eg:-

Consider :-

```
static int fun(int n) {
    if (n==0)
        return;

    sout(n)
    fun(n--);
}
```

Here, it does not print n to 1 numbers, as n-- passes value of n first and then subtract n.

So, initially if $n = 4$.
It passes 4 to next call
and then makes n as 3.

But $n = 4$, has again been passed to next call, so the next function call also again passes $n = 4$ and then makes it 3.

Stack Overflow Occurs!

output

$f(4) \rightarrow 4$

$\downarrow$

$f(4) \rightarrow 4$

$\downarrow$

$f(4) \rightarrow 4$

Calls same n value function again and again.

NOTE:- So better use $f(--n)$
So, n is subtracted first and then passed.

for example:- Initially n = 5,
It makes n=3, and calls f(3)

Code:-

```
public class Concept {
    public static void main(String[] args) {
        fun(n:5);
    }
    static void fun(int n) {
        if (n == 0) {
            return;
        }
        System.out.println(n);
//          fun(n--);
        fun(--n);
//          n-- vs --n
    }
}
```

---

Q). Reverse of a number:-

Eg:- N = 1824 => 4281

Logic:-

$4 \times 10^3 + 182$

$2 \times 10^2 + 18$

$1 \times 10 + 8$.

$= 4281$

Logic :-

rem = N % 10.

return rem * (int)(Math.pow(10, digits-1)) + helper(n/10, digits-1).

Code :-

```
    int digits = (int)(Math.log10(n)) + 1;
    return helper(n, digits);
}

private static int helper(int n, int digits) {
    if (n%10 == n) {
        return n;
    }
    int rem = n % 10;
    return rem * (int)(Math.pow(10, digits-1)) + helper(n:
}

public static void main(String[] args) {
    rev1(n:1234);
    System.out.println(sum);
```

Q). Is the given number a palindrome.

Soln:- $N_1 = 121$ Palindrome.

$N_2 = 121$

Logic:- First find reverse of that number by using above recursion. and pass it and original number to a function to check whether it is equal or not.

Code:-

```
private static int helper(int n, int digits) {
    if (n%10 == n) {
        return n;
    }
    int rem = n % 10;
    return rem * (int)(Math.pow(10, digits-1)) + helper( n: n/10,
}

static boolean palin(int n) {
    return n == rev(n);
}

public static void main(String[] args) {
    System.out.println(rev( n: 1234321));
}
```

→ checking whether it contains single digit.

Q1. Count no. of zeroes in a number

Soln:- N = 1002

Ans = 2

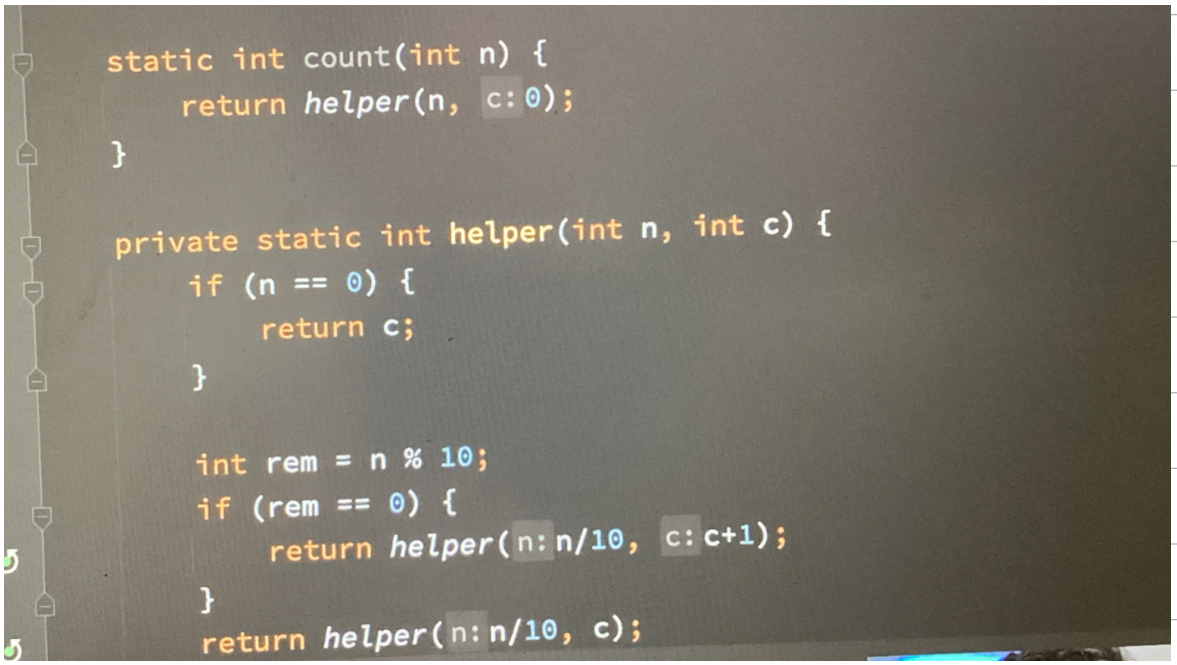
Logic:- When we get remainder as 0 increment counter by 1.

```
F(N, count){
if (N==0)
        return count;
else{
    rem = N%10;
if (rem==0)
        F(N/10, c+1);
```

else

f(N/10, c);

}.



```
static int count(int n) {
    return helper(n, c: 0);
}

private static int helper(int n, int c) {
    if (n == 0) {
        return c;
    }

    int rem = n % 10;
    if (rem == 0) {
        return helper(n: n/10, c: c+1);
    }
    return helper(n: n/10, c);
}
```