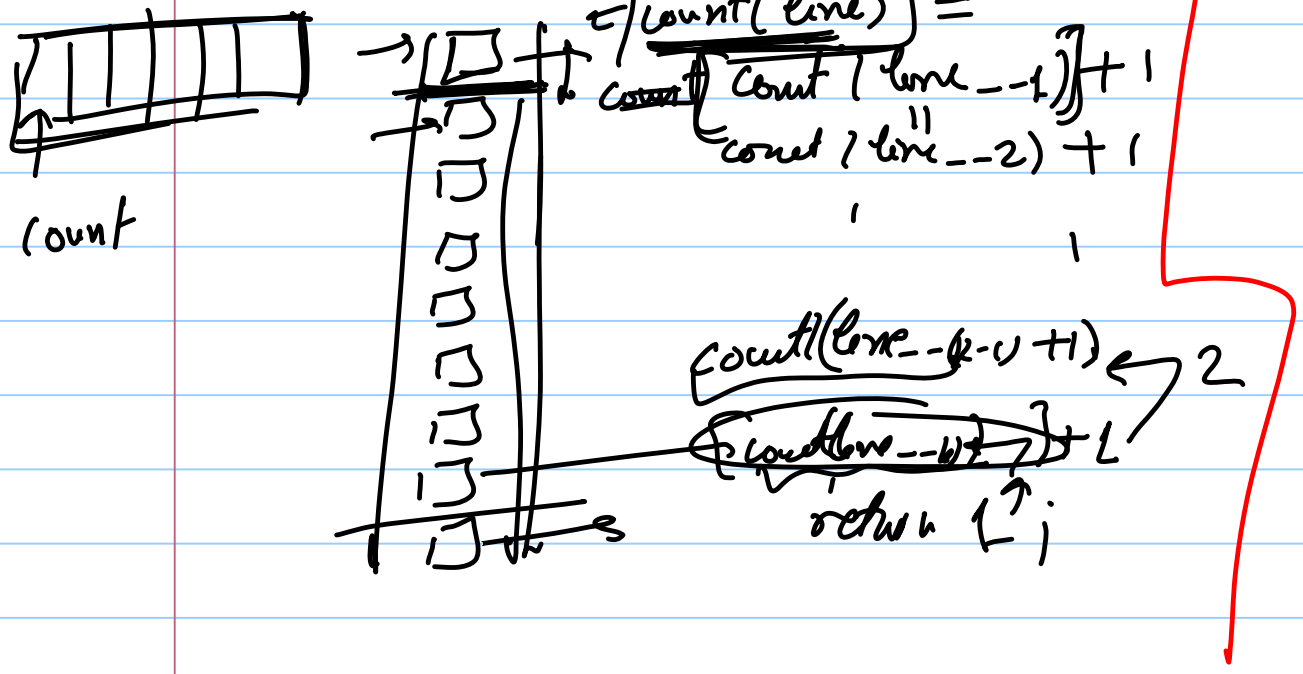
Note Title 29-10-2018

Oct 29 2018

Recursion



"Correctness"

Characteristics:-

(1) Self similar
simpler to solve or smaller
+ something in size. & more

count(line)
= count(line--1) + 1
↓
simpler problems

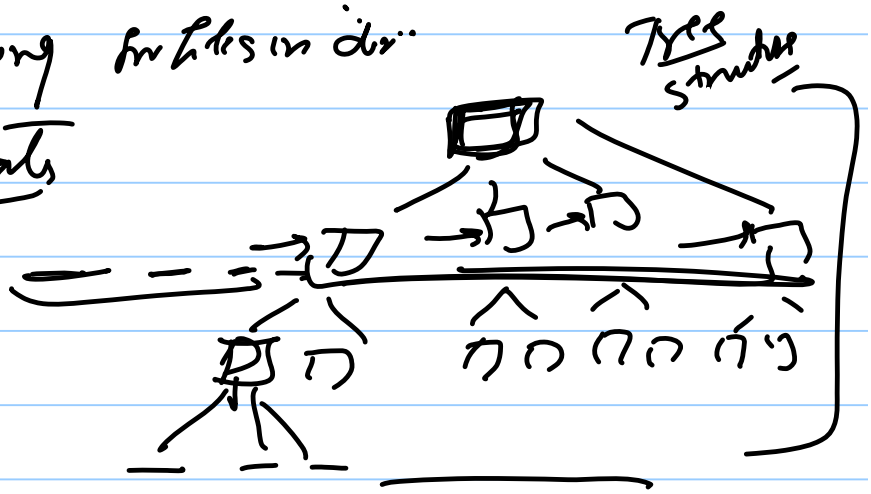
(2) Base Case:- Nothing but a "self similar" problem that you can solve directly

Example of where we use Recursion:

(1) Looking for word in dictionary

(2) Looking for files in dir.

(3) Fractals



Prolog → PL:- No Loops At all

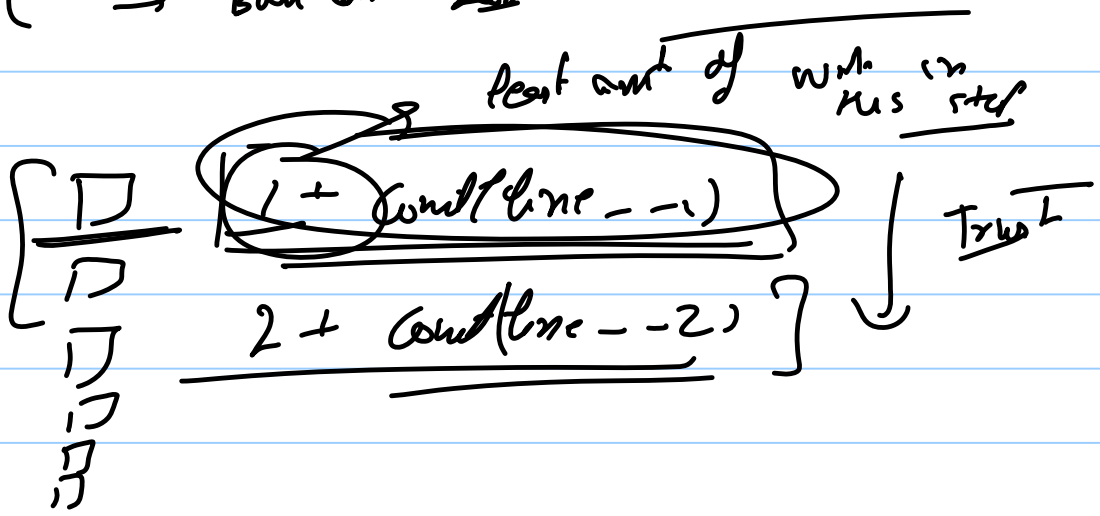
↳ only recursion

→ When / when ~~you~~ would you use recursion.

"Smart" about how to decompose into smaller problems

(1) You can decompose the problem into "simpler" smaller self. similar problems + Δ

(2) Look for "Base case"



3 Rules for recursion

1. → Every input has a case { Recursive Case / Base Case
2. → There "must" be a base case(s) & should be reachable from every input.
3. → The recursive case must make the problem simpler & progress towards the base case(s)

```
recursiveFunc() {
    if (test for simple/base case(s)) {
        Compute the sol. directly // No recursion
    } else {
        1. Break the problem into smaller subproblem(s) of same type
        2. call recursiveFunc on each of smaller problems.
        3. Reassemble the results of smaller problems
    }
}
```

Iterative Program using loops. $n \geq 1$

```
int fact(int n) {
    int result = 1;
    for(int i = 1; i <= n; i++) {
        result = result * i;
    }
    return result;
}
// n >= 1
```

$1! = 1$

$2! = 2 \times 1$

$3! = 3 \times 2 \times 1$

$n! = n \times (n-1) \times (n-2) \times \dots \times 1$

$\Rightarrow n! = (n) \times (n-1)!$

→ Recursive Program No Loops

```
int fact(int n) {
    if(n == 1) {
        return 1;
    } else {
        return n * fact(n-1);
    }
}
```

// Base Case

fact(4);

→ Recursive trace:

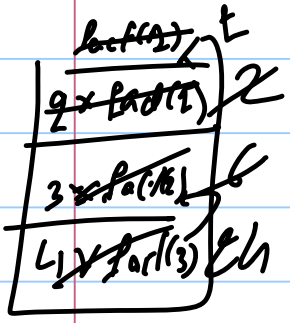
fact(4) ⇒ 4 * fact(3) ← 24

3 * fact(2) ← 6

2 * fact(1) ← 2

1 ← 1

Base Case



Iterative:

```
int fact(int n){
  if(n == 1){
    return 1;
  }else{
    return n * fact(n-1);
  }
}
```

// n >= 0

```
int mystery(int n) {
    if(n < 10) {
        return n;          // single digit
    } else {
        int a = n/10;
        int b = n%10;
        return mystery(a+b);
    }
}
```