Psuedocode:
It is one of the ways to solve a computing problem.
It uses normal everyday english text to define the problem.
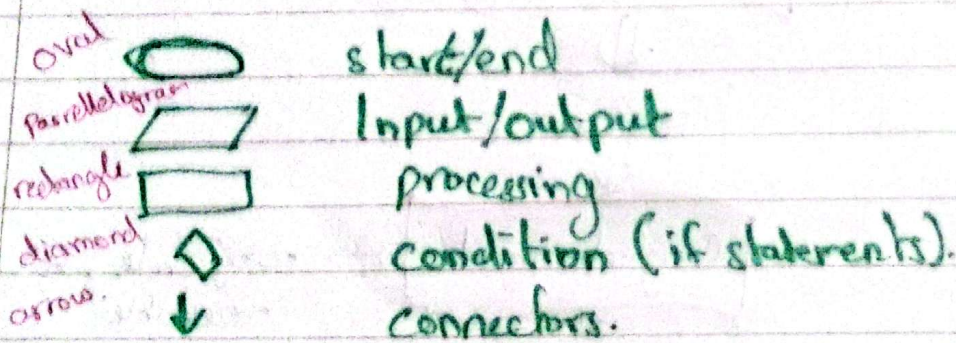4 Steps:

1) Purpose   what is the problem?
2) Input   What inputs do we need?
3) ~~Output~~ Process   How to process the inputs?
4) Output   Display the result.

Flow charts:
~~this~~ In Flow charts, we make a chart using the
4 methods shown in Psuedocode.

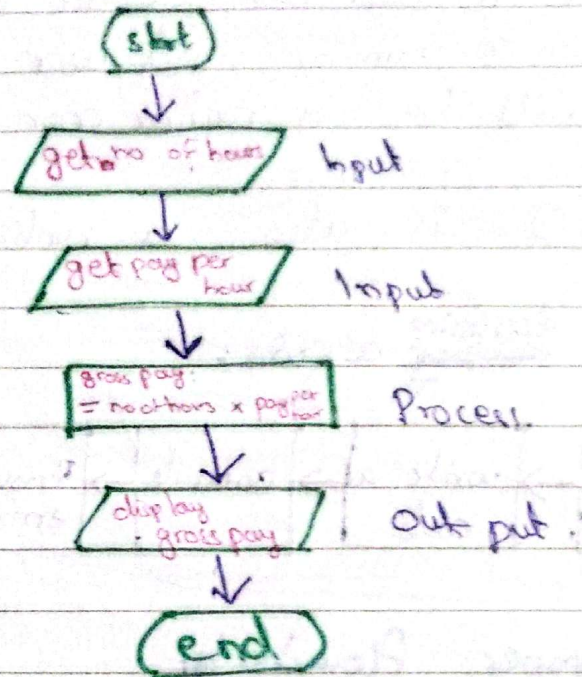~~Example~~ Format of a flow chart:

oval   start/end
parallelogram   Input/output
rectangle   processing
diamond   condition (if statements).
arrow   connectors.

example:
We need to calculate the gross pay of a person.

```
                    ( start )
                        │
                        ▼
          ┌─────────────────────┐
         /  get no of hours      /    Input
          └─────────────────────┘
                        │
                        ▼
          ┌─────────────────────┐
         /  get pay per          /    Input
        /     hour               /
          └─────────────────────┘
                        │
                        ▼
          ┌─────────────────────┐
          │  gross pay:          │
          │ = no of hours × pay  │    Process
          │            per hour  │
          └─────────────────────┘
                        │
                        ▼
          ┌─────────────────────┐
         /  display              /    Output :
        /     gross pay          /
          └─────────────────────┘
                        │
                        ▼
                    ( end )
```

Errors:

1) **Syntax error:**
   This error occurs when there is a problem in the code and this stops the compiler from compiling.

2) **logical error:**
   This error is not shown by the IDE. Instead it ~~~~ takes place when the code is correct but output is wrong.
   e.g. we wanted a + b in code but accidently wrote a * b, the code works but output wrong.

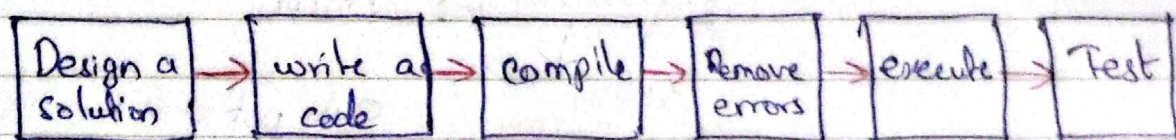## Runtime error:

This error takes place while the code is being run.
e.g Input is asked from the user and the Input should
be a number and user types a string, so
it will be a runtime error.

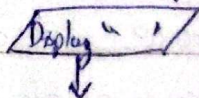- $1 \div 0$ is also a runtime error.

Cycle of ~~running~~ designing a code:

```
┌──────────┐   ┌─────────┐   ┌─────────┐   ┌────────┐   ┌────────┐   ┌──────┐
│ Design a │ → │ write a │ → │ compile │ → │ Remove │ → │execute │ → │ Test │
│ solution │   │  code   │   │         │   │ errors │   │        │   │      │
└──────────┘   └─────────┘   └─────────┘   └────────┘   └────────┘   └──────┘
```

Writing proper flowcharts:

tip 1) Write the display message in inverted commas
and the process without inverted commas (" ").

ie   /Display " "/
                 ↓
           /Input a number/

tip 2) write the display message and input as two
seperate boxes.

tip 3) Use variables.
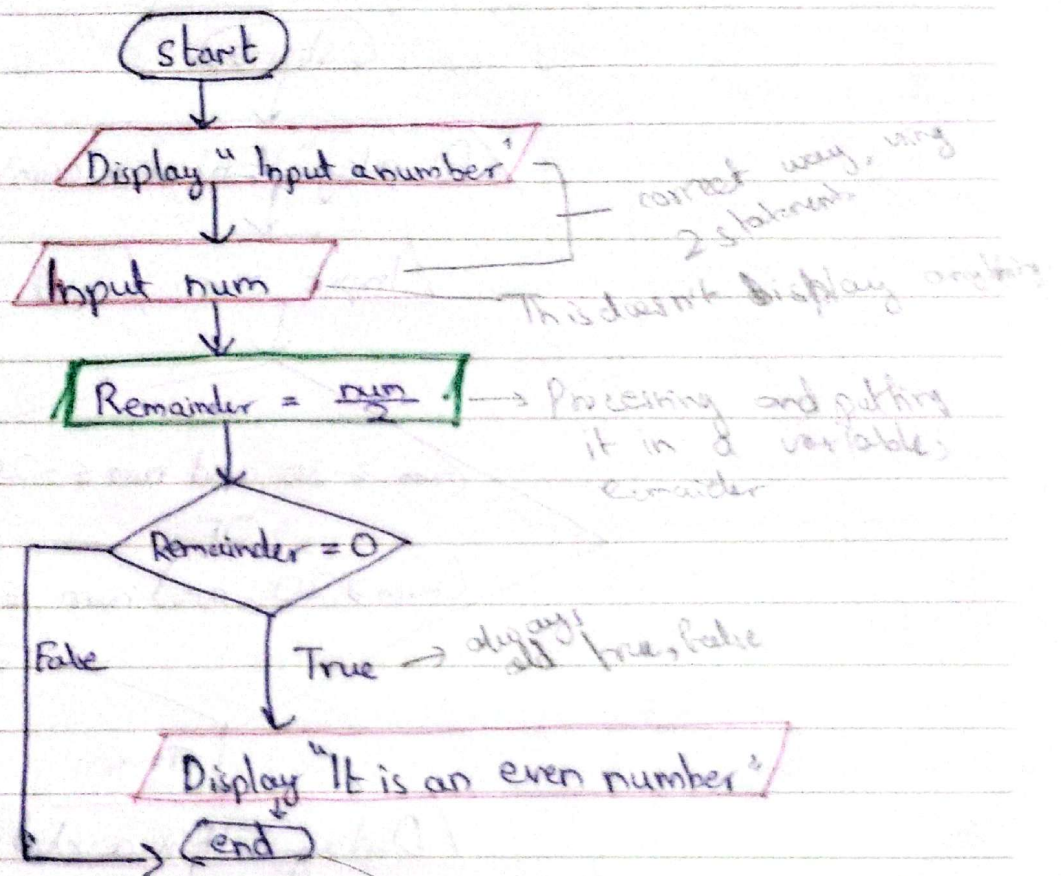
tip 4) Don't need to write display. just write in
" ".

Examples:

Q1) Design a flowchart that displays True if a number is an even number.

start

Display " Input a number"

Input num

Remainder = $\frac{num}{2}$

Remainder = 0

False

True → always true, false

Display "It is an even number"

end

— correct way, using 2 statements

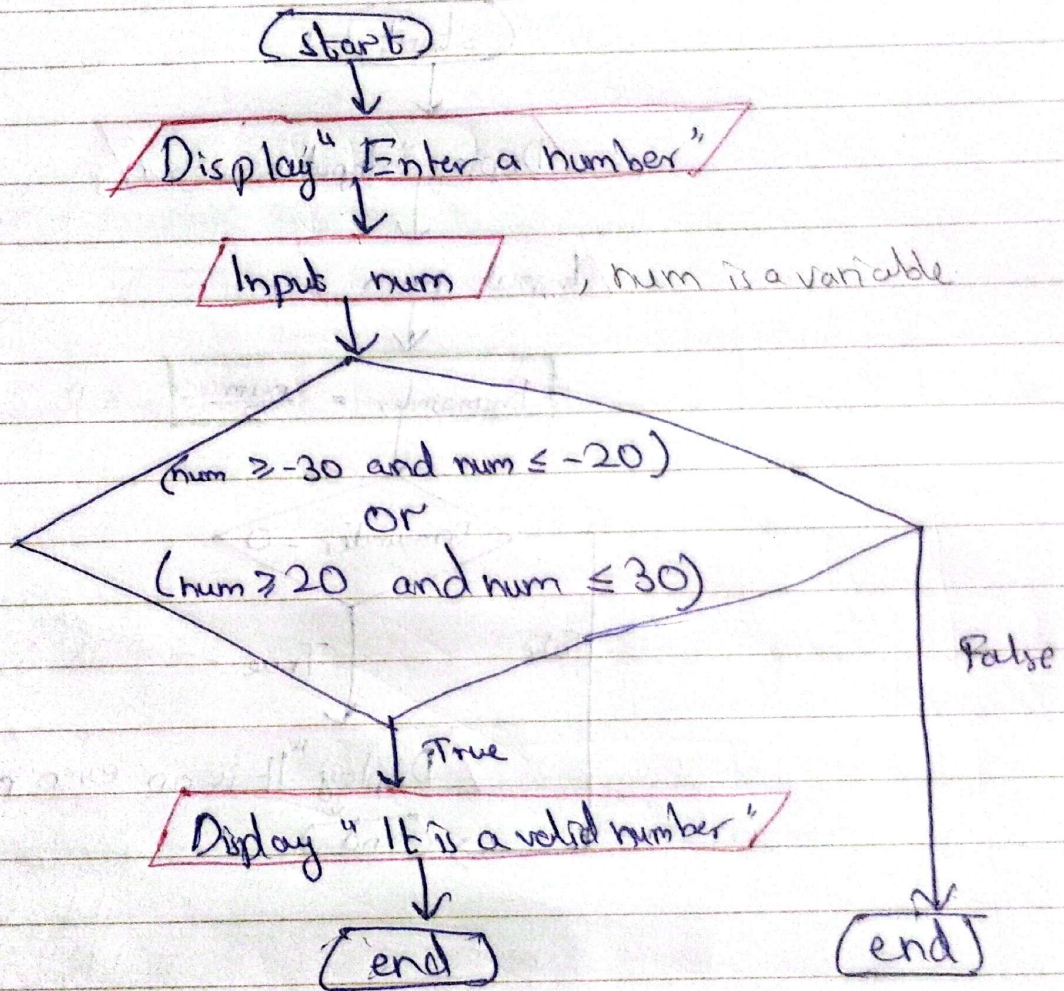This doesn't display anything

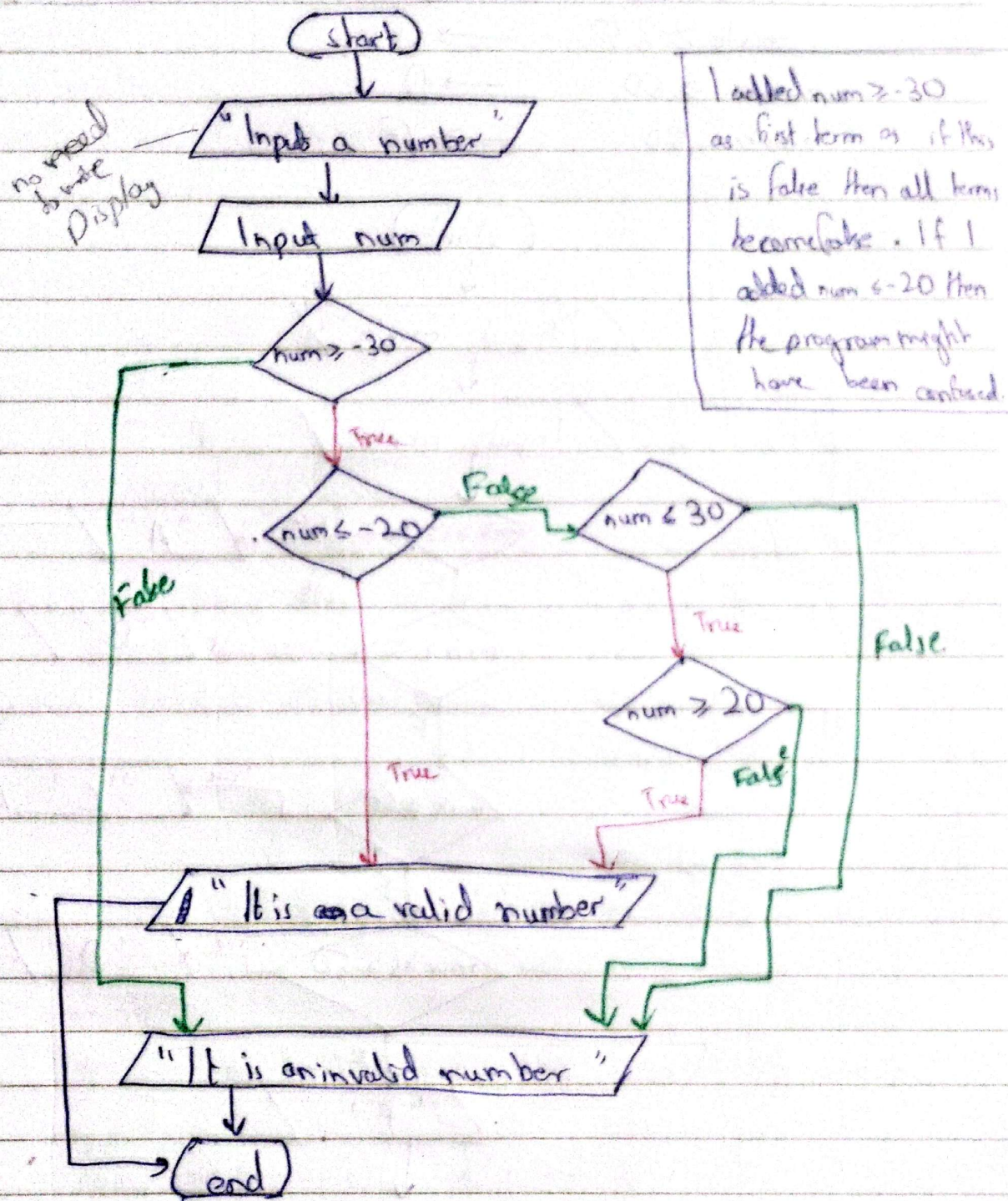→ Processing and putting it in a variable remainder

I could have also used multiple end statements

Q2) Display "it is a valid no" if the number entered
by user is between -20 and -30 or
between 20 and 30.

Start

Display " Enter a number "

Input num   , num is a variable

$$(num \geq -30 \text{ and } num \leq -20)$$
$$or$$
$$(num \geq 20 \text{ and } num \leq 30)$$

False

True

Display " It is a valid number "

end   end

Q.#3 Same previous question but without using and, or. Also add "It is an invalid number".

Start

"Input a number"

Input num

no need of write Display

num ≥ -30

I added num ≥ -30 as first term as if this is false then all terms become false. If I added num ≤ -20 then the program might have been confused.

True

num ≤ -20 → False → num ≤ 30

False

False

True

num ≥ 20

True   False

"It is a valid number"

"It is an invalid number"

end

Yousaf

(Q # 4)

marks ≥ 85 ———→ A

mark ≥ 75 ———→ B

mark ≥ 65 ———→ C

marks ≥ 50 ———→ D

marks < 50 ———→ F

write a program for this.

```
          ( start )
             ↓
   / "Input marks" /
             ↓
   / Inputs markss /
             ↓
      < marks ≥ 85 > ——True——→ / "A" / ———→
             ↓ F                              ↓
      < marks ≥ 75 > ——True——→ / "B" / ——→  ( end )
             ↓ F                              ↑
      < marks ≥ 65 > ——True——→ / "C" / ——→
             ↓ F                              ↑
      < marks ≥ 50 > ——True——→ / "D" / ——→
             ↓ F
         / "F" /
             ↓
          ( end )
```