Date : 2nd Sep '2021
Lecture : 1st

mail id : sumeet.malik@pepcoding.com
Group mail id: fjp1@pepcoding.com

Language
- Print or Output
- Variable
- Conditionals
- Loops
- Taking Input

] Need to Know

How to print in Java

```
System.out.println("Hello World");
```
→ Prints to the console and then moves to next line as well.

```
System.out.print("Hello World");
```
→ Prints to the Console.

↳ Shortcut :
Sysout + ctrl + Space

ABSTRACTION

('\n') → can also be used to move to the next line

@ Print Z

Output :

★ ★ ★ ★ ★
  ★
 ★
★
★ ★ ★ ★ ★

Concept :

★ ★ ★ ★ ★
_ _ _ ★
_ _ ★
_ ★
★ ★ ★ ★ ★

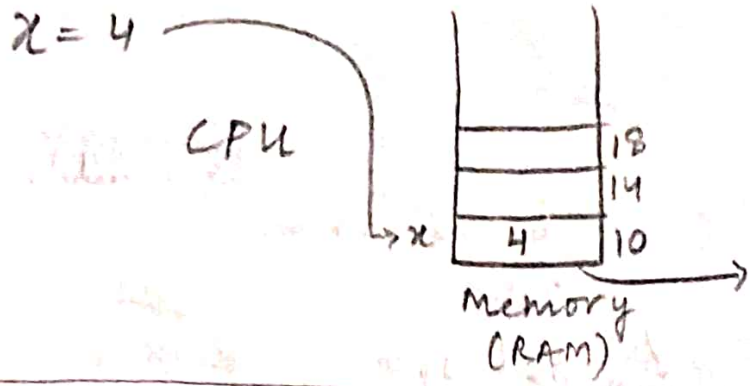
→ Code :

```
class Print Z {
public static void main(String[] args) {
    System.out.println("★★★★★ ");
    System.out.println("_ _ _ ★");
    System.out.println("_ _ ★");
    System.out.println("_ ★");
    System.out.println("★★★★★");
    System.out.println("ALTER");
    System.out.print("★★★★★\n_ _ _ ★_\n_ _ ★_ _\n_ ★_ _ _\n★★★★★\n");
    ;
}
}
```

Code1 = the six println lines
Code2 = the print line

VARIABLES

CODENATION TOWER RESEARCH

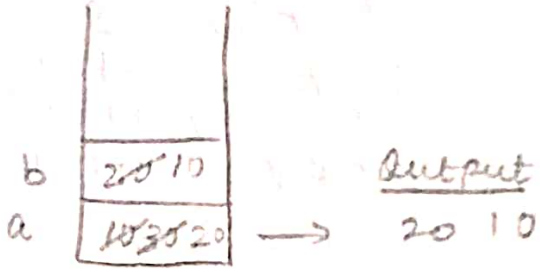



```
int x = 4;
System.out.println(x);
```

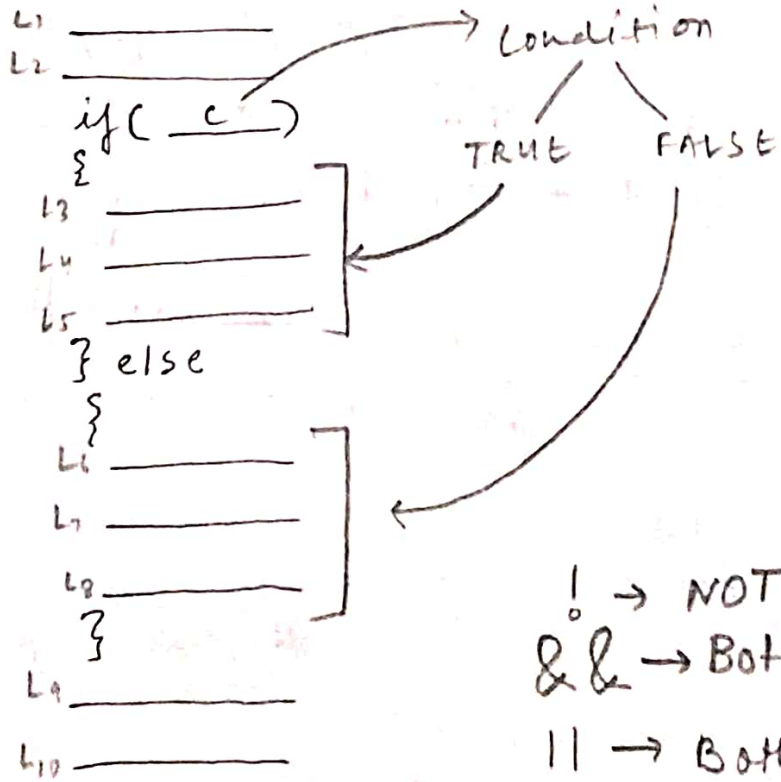
Date: 3rd Sept'2021
lecture: 2nd

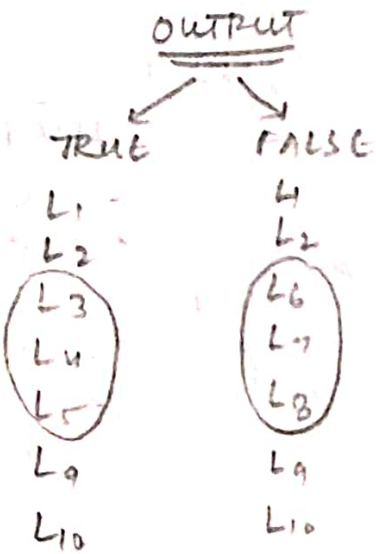
Output ?

```
int a = 10;
int b = 20;
a = a + b;   10+20 = 30
b = a - b;   30-20 = 10
a = a - b;   30-10 = 20
System.out.println(a + " " + b);
```



⑤ Conditional Statements





! → NOT

&& → Both True → TRUE

|| → Both False → FALSE

= → Assign

== → Equal (check)

GRADING SYSTEM

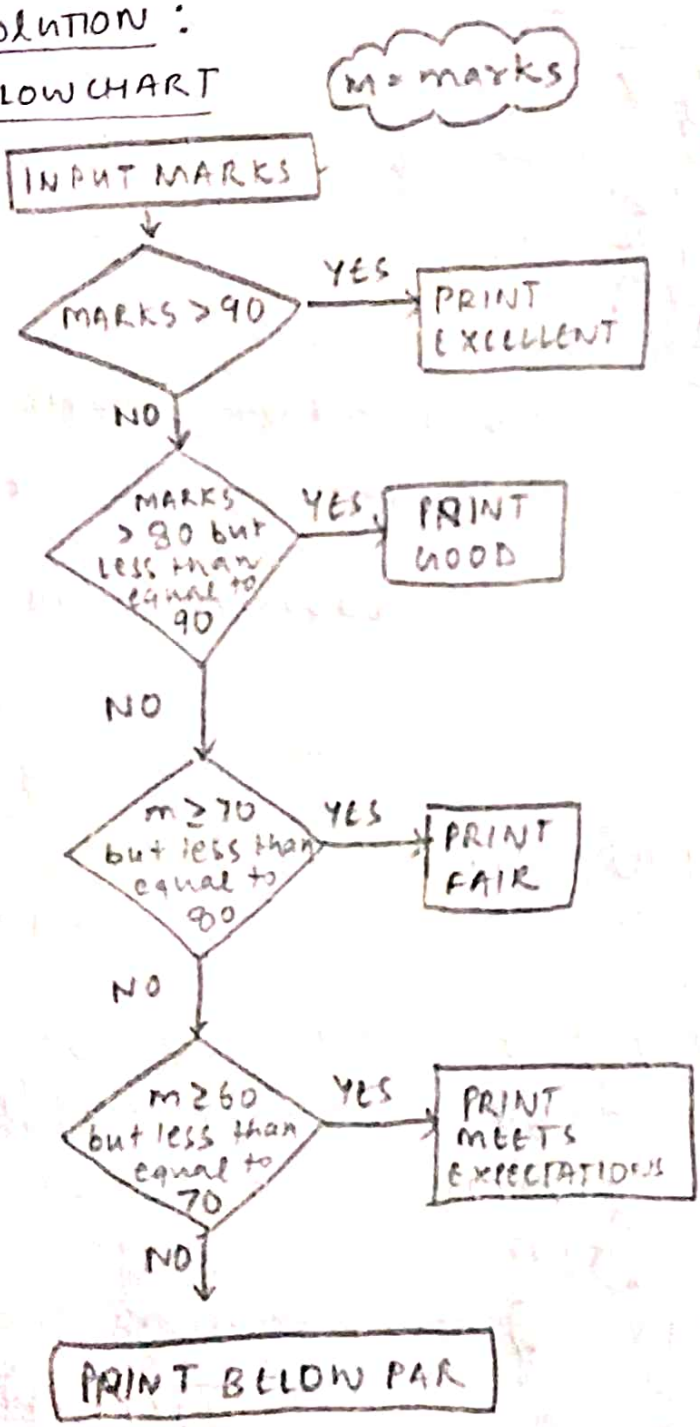
ELSE IF LADDER

Question:

MARKS
- (m > 90) ⟶ EXCELLENT
- (90 ≥ m > 80) ⟶ GOOD
- (80 ≥ m > 70) ⟶ FAIR
- (70 ≥ m > 60) ⟶ MEETS EXPECTATIONS
- (m ≤ 60) ⟶ BELOW PAR

Solution:

FLOWCHART



Code

```
public static void main (String [] args)
{ Scanner s = new Scanner (System.in);
  int marks = s.nextInt();
if (marks > 90){
System.out.println ("EXCELLENT");
}
else if (marks > 80){
System.out.println("GOOD");
}
else if (marks > 70){
System.out.println ("FAIR");
}
else if (marks > 60){
System.out.println ("MEET EXPECTATIONS");
}
 else {
  System.out.println ("BELOW PAR");
  }
}
```

Output

98 ⟶ EXCELLENT
86 ⟶ GOOD
77 ⟶ FAIR
64 ⟶ MEET EXPECTATIONS
55 ⟶ BELOW PAR

+ ↓ 8+5 = 13

− ↓ 8−5 = 3

* ↓ 8*5 = 40

/ ↓ 8/5 = 1 (QUOTIENT)

% ↓ 8%5 = 3 (REMAINDER)

Odd | EVEN

```
public static void main(String[] args) {
 Scanner s = new Scanner(System.in);
  int marks = s.nextInt(); → 58, 57
 if (marks % 2 == 0) {
    System.out.println("EVEN");
 }
 else {
   System.out.println("ODD");
 }
}
```

57%2 ≠ 0

58%2 = 0

CONSOLE

EVEN

ODD

PLUS CONCEPT

3

NUMBERS ← 3K+0 → 18, 12

NUMBERS ← 3K+1 → 19, 7

NUMBERS ← 3K+2 → 20, 14

18 = 3·6

12 = 3·4

19 = 3·6+1

7 = 3·2+1

20 = 3·6+2

14 = 3·4+2

CONSOLE

3K+2

3K+1

Code

```
public static void main(String[] args)
{Scanner s = new Scanner(System.in);
 int marks = s.nextInt();
 if (marks % 3 == 0) {
 System.out.println("3K");
 }
 else if (marks % 3 == 1) {
 System.out.println("3K+1");
 }
 else {
 System.out.println("3K+2");
 }
}
```

7

8

LOOPS

- ARE USED TO PERFORM BLOCK OF STATEMENTS REPEATEDLY.
- ARE EXECUTED TILL THE CONDITION HOLDS TRUE.

QUESTION: PRINT NO. 1 to 10.

```
int i = 1;→INITIALIZATION (1)
while (i<=10)→CONDITION (2)
{
   →STATEMENT (3)
System.out.println(i);
i++;→UPDATION (4)
} //WHILE LOOP
```

ORDER: (1) → (2) → (3) → (4)

```
for(int i=1; i<=10; i++)
{
   → (3)
System.out.println(i);
} // FOR LOOP
```

(1) (2) (4)

ORDER: (1) → (2) → (3) → (4)

OUTPUT

1
2
3
4
5
6
7
8
9
10

```
int i=1;→(1)
do {
   → (2)
System.out.println(i);
i++;→(3)
}
while(i<=10);→(4)
//DO-WHILE LOOP
```

ORDER: (1) → (2) → (3) → (4)

INFINITE LOOP

- FOR
  - `for( ; ; )`
  - `for(x; y; z++);` → EMPTY BODY
  - `for( ; true; )`
- WHILE OR DO-WHILE
  - `while(true) { ==== }`
  - `while(i<10);` → EMPTY BODY
  - `while(i<10) {;}`

WHERE TO USE WHICH LOOP

RULE:

WHEN → INITIALIZATION, CONDITION CHECK, UPDATION → FIXED → **FOR LOOP**

→ HAVE → COMPLEX LOGIC TO MODIFY LOOP MODIFICATION VARIABLE → **WHILE LOOP**

→ WE CAN WRITE BETTER INFINITE LOOPS.

UPDATION OPERATORS

→ POST, PRE — INCREMENT OPERATOR, DECREMENT OPERATOR

∴ PRE OPERATORS (UPDATION) → FIRST UPDATE THE VALUE THEN USE IT

`++i` → PRE-INCREMENT — EQUIVALENT TO → $i = i + 1$

`--i` → PRE-DECREMENT — EQUIVALENT TO → $i = i - 1$

∴ POST OPERATORS (UPDATION) → UPDATE THE VALUE BUT USES THE ORIGINAL VALUE

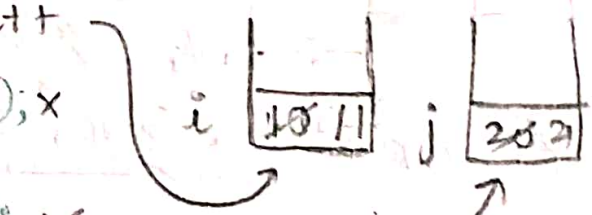
`i++` → POST-INCREMENT

`i--` → POST-DECREMENT

```
int i=10; → 10
if(i++==j) → 11 ≠ 10
System.out.println(i+"IS GOOD");X
else
System.out.println(i+"IS BAD");✓
int j=20; → 20
if(++j==j) → 21==21
System.out.println(j+"IS GOOD")✓
else
System.out.println(j+"IS BAD");
```

i++ changes the value of i to 11 but returns old value of i = 10

++j changes the value to 21 and returns new value



OUTPUT

11 IS BAD 21 IS GOOD

⑧ USER TAKING INPUT

```
import java.util.Scanner; → HEADER FILE
Scanner sc = new Scanner(System.in);
int n = sc.nextInt();
String name = sc.nextLine();
int n = Integer.parseInt(sc.nextLine());
```

↳ THIS STATEMENT IS USED TO CONVERT THE N INTO INTEGER FORM.

READING FROM KEYBOARD AND TAKING THE VALUE TO RAM

THIS LINE CREATES A SCANNER

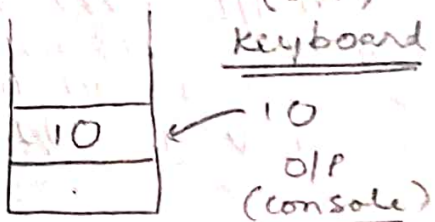
```
Scanner scn = new Scanner(System.in);
```

Scanner → TYPE; scn → variable; System.in → input stream that gives stream of bytes

```
int i = scn.nextInt();
```

↳ THIS LINE READS FROM KEYBOARD

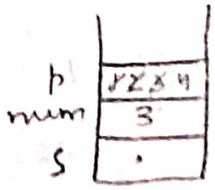
```
Scanner s = new Scanner(System.in);
int n = s.nextInt(); → 10
System.out.println("I ENTERED" + n); → I ENTERED 10
```



"I ENTERED" → SOMETHING IN DOUBLE QUOTE IS STRING; n → 10

```
Scanner s = new Scanner(System.in);
int num = s.nextInt(); → (3)
int p = 1; p=1
while (p <= num) {
    System.out.println(p); → (1) (2)
    p++; → (2) (3) (4)
}
System.out.println("DONE");
```

↳ DONE



Input	Output
3	1 2 3 DONE

HOMEWORK

OUTPUT

1 is ODD ✓
2 is EVEN ✓
3 is ODD ✓
4 is EVEN ✓
5 is ODD ✓
⋮ ⋮

INPUT

5

```
Scanner s = new Scanner(System.in);
int n = s.nextInt(); → 5
for (int i=1; i<=n; i++)
{
  if(i%2==0) { f t f t f
  System.out.println(i+" is EVEN");
  } else t t t
  {
    System.out.println(i+" is ODD");
  }
}
```

t t.

1<=5 2<=5 3<=5 4<=5 5<=5

Date: 4th Sep '2021
Lecture: 3rd

PRIME NUMBERS LIKE LIFE. THEY ARE VERY LOGICAL BUT YOU CAN NEVER WORK OUT THE RULES, EVEN IF YOU SPENT ALL YOUR TIME THINKING ABOUT THEM.

IS A NUMBER PRIME

PRIME NUMBER → DIVISIBLE BY 1 AND ITSELF

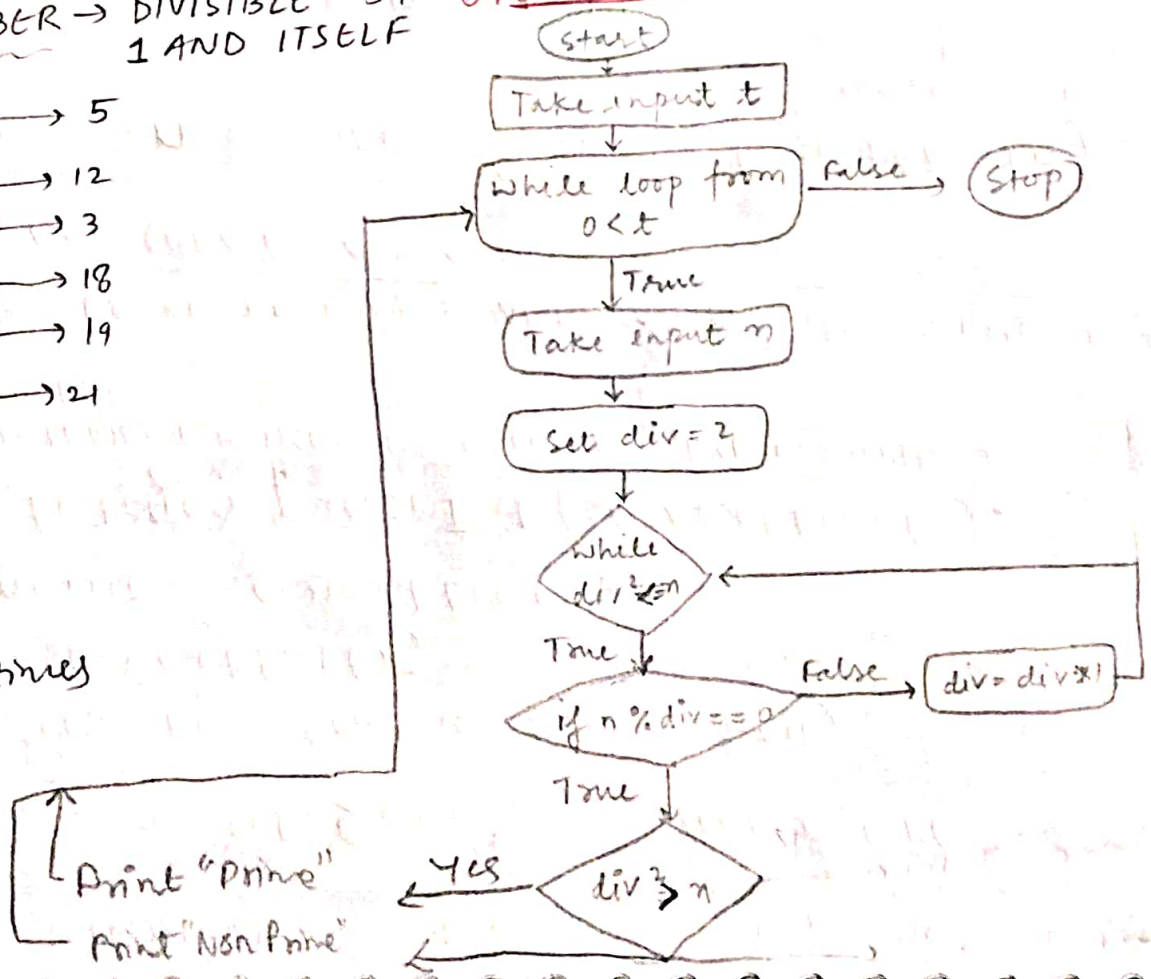
Ques: t → 5

t times:
- n → 12
- n → 3
- n → 18
- n → 19
- n → 21

OUTPUT

prime
not prime
not prime
... t no. of times

FLOWCHART



```
public static void main (String [] args) {
  Scanner sc = new Scanner (System.in);
    int t = sc.nextInt();
  for (int i = 0; i < t ; i++);
  {
    int n = sc.nextInt();
    int div = 2;
    while (div * div <= n)
    {
      if (n % div == 0) {
        break;
      }
      div++;
    }
    if (div * div > n) {
      System.out.println ("PRIME");
    } else {
      System.out.println ("NOT PRIME");
    }
  }
}
```

t = 5

n ⟶
- 23 → PRIME
- 34 → NOT PRIME
- 23 → PRIME
- 45 → NOT PRIME
- 56 → NOT PRIME

2 x 2 = 4
3 x 3 = 9
4 x 4 = 16

n = 53

isPrime	div	r
(t)	2	1
	3	2
	4	1
	5	3
	6	5
	7	4
	8	

```
boolean isPrime = true;
int div = 2;
while (div * div <= n){
  int r = n % div;
  if (r == 0){
    isPrime = false;
    break;
  }
  div++;
}
```

Approaches

[1 to n]

[2 to n-1]

[2 to n/2]

[2 to $\sqrt{n}$] ⟶ Best One

⑧ PRINT ALL PRIMES TILL (N) (H.W)

```
public static void main (String [] args)
{
Scanner scn = new Scanner(System.in);
   int low = scn.nextInt();  ⟶ 7
   int high = scn.nextInt(); ⟶ 13
for (int n = low ; n <= high ; n++)
   {
   int count = 0;
   for (int div = 2; div * div <= n; div++)
   {  if (n % div == 0) {
         count++;
         break;
      }
   }
   if (count == 0) {
   System.out.println(n);
   }
  }
}
```

l	h	n	div	count
7	13	7 8 9 10 11	2	0 0 0 0 1 1 1

console

```
7
13

7
11
13
```

ALTER

```
{
  boolean isPrime = true;
  int div = 2;
  while (div * div <= n) {
    int rem = n % div;
    if(rem == 0) {
       isPrime = false;
       break;
    }
    div++;
  }
  if (isPrime == true){
   System.out.println(n); }
```

PRINT FIBONACCI NUMBERS TILL N

"OUR LIFE IS LIKE FIBONACCI NUMBERS. EVERYTHING THAT HAPPENED IN THE PAST, IS ADDED TO THE PRESENT, TO MAKE THE FUTURE"

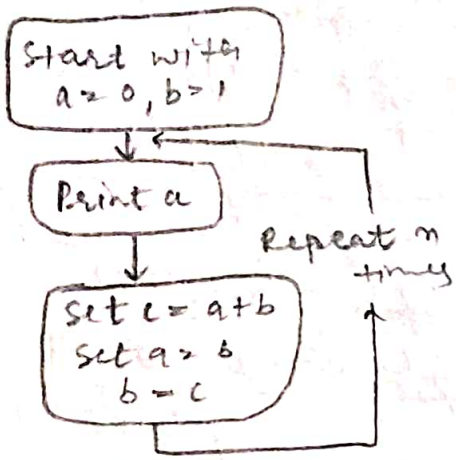
1 2 3 5 8 . . .

0+1 1+1=2 1+2=3 2+3=5 3+5=8

$\therefore F(0) = 0, \quad F(1) = 1$

$$F(n) = F(n-1) + F(n-2)$$

FLOWCHART




```
public static void main (String [] args) {
Scanner sc = new Scanner (System.in);
System.out.println ("FIBONACCI NUMBERS");
 int n = sc.nextInt(); ⟶ 10
 int a = 0;
 int b = 1;
for (int i=0; i<n; i++)
{
 System.out.println(a);
 int c = a+b;
  a = b;
  b = c;
 }
}
```

output

10

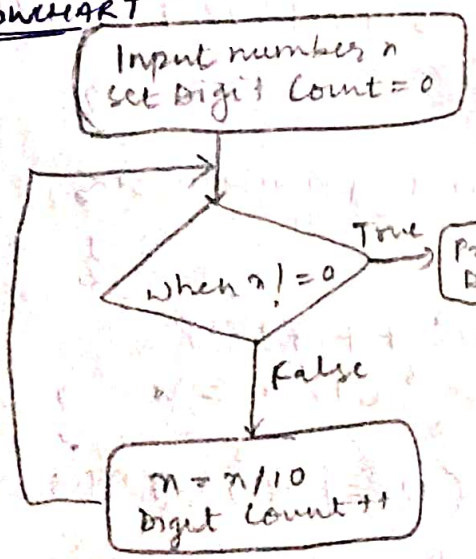
0 1 2 3 5 8 13 21 34

Date: 5th Sept '2021
Lecture: 4th

"NUMBERS IS THE RULER OF FORMS AND IDEAS, AND THE CAUSE OF GODS AND DEMONS"

COUNT DIGITS IN A NUMBER

FLOWCHART



AGAR CONSTRAINT ME 10^ HAI TO INT USE KRANA HAI;

"EVERY DIVISION STRIPS OUT THE LAST DIGIT."

GIVEN NO: 9543

10	9543	Rem.	Digit Count
10	954	3	1
10	95	4	2
10	9	5	3
	0	9	4

Divided 4 times by 10 or 10000

⟶ No. of digits is 4

```
public static void main (String [] args) {
 Scanner s = new Scanner (System.in);
  int n = s.nextInt (); ⟶ 2483
System.out.println ("COUNT DIGITS IN A NUMBER");
int digit = 0;
while (n != 0) ✓
{   n = n/10; ✓
    digit ++; ✓ }
}
System.out.println ("NO. OF DIGITS ARE : " + digit);
}
```

10	2483 = n	%	count
10	248	3	1
10	24	8	2
10	2	4	3
	0	2	4

(4)

NOTE: QUESTION'S KE CONSTRAINTS DHYAANS SE PADHEY

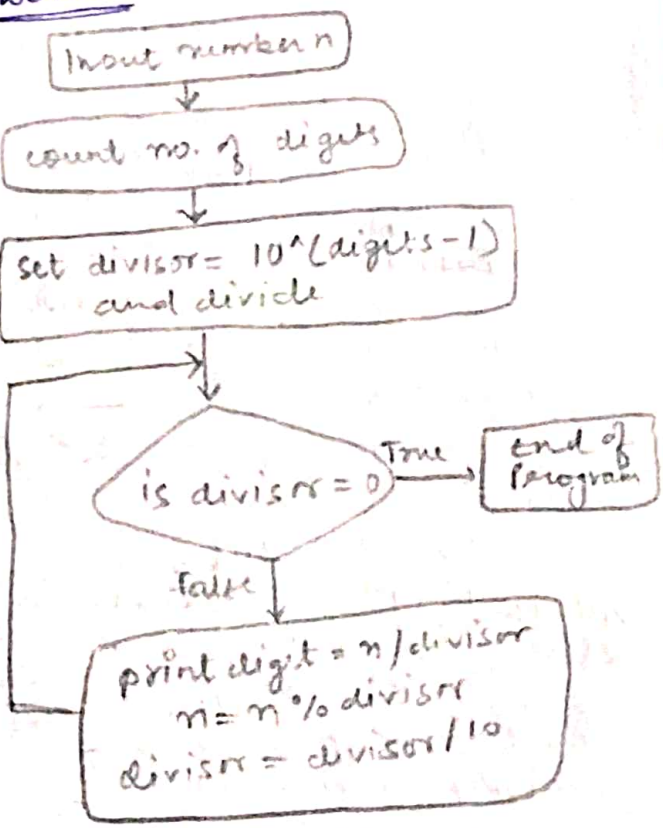
DIGITS OF A NUMBER

NUMBERS CONSTITUTE THE ONLY UNIVERSAL LANGUAGE. THEY RULE THE UNIVERSE

divisor	Remainder / Quotient	Extracted digit
10000	23475	
1000	(2) → 3475	(2)
100	(3) → 475	(3)
10	(4) → 75	(4)
1	(7) → 5	(7)
	(5) → 0	(5)

FIRST DIVISOR = 10 ^ (NUMBER OF DIGITS − 1)

Flowchart



```
int div = 1;
int temp = n;
while (temp >= 10) {
    temp = temp / 10;
    div *= 10;
}
```

	n	Div.
10	2347	1
10	234	10
10	23	100
	2	1000

```
while (div >= 1) {
    int q = n / div;
    int r = n % div;
System.out.println(q);
    n = r;
    div = div / 10;
}
```

1000	2347^n
100	$2 - 347^n$
10	$3 - 47^n$
1	$4 - 7^n$
	$7 - 0^n$

// Code (Final)

```
public static void main (String [] args) {
Scanner sc = new Scanner (System.in);
    int n = sc.nextInt();
    int div = 1;
    int temp = n;
    while (temp >= 10) {
        temp = temp / 10;
        div *= 10;
    }
    while (div >= 1) {
        int q = n / div;
        int r = n % div;
    System.out.println(q);
        n = r;
        div = div / 10;
    }
}
```

⑧ INVERSE OF A NUMBER

EX 1

⑥	⑤	④	③	②	①	← POSITION INDEX
4	2	6	1	3	5	← GIVEN NUMBER

⇓

4	1	6	2	5	3	← POSITION INDEX
⑥	⑤	④	③	②	①	← INVERTED NUMBER

EX 2

3	5	2	1	4
⑤	④	③	②	①
10^4	10^3	10^2	10^1	10^0

4	1	5	3	2
⑤	④	③	②	①
10^4	10^3	10^2	10^1	10^0

10	3 5 2 1 4	r
10	3 5 2 1	4
10	3 5 2	1
10	3 5	2
10	3	5
	0	3

POSITION		DIGIT
①	⟶	4
②	⟶	1
③	⟶	2
④	⟶	5
⑤	⟶	3

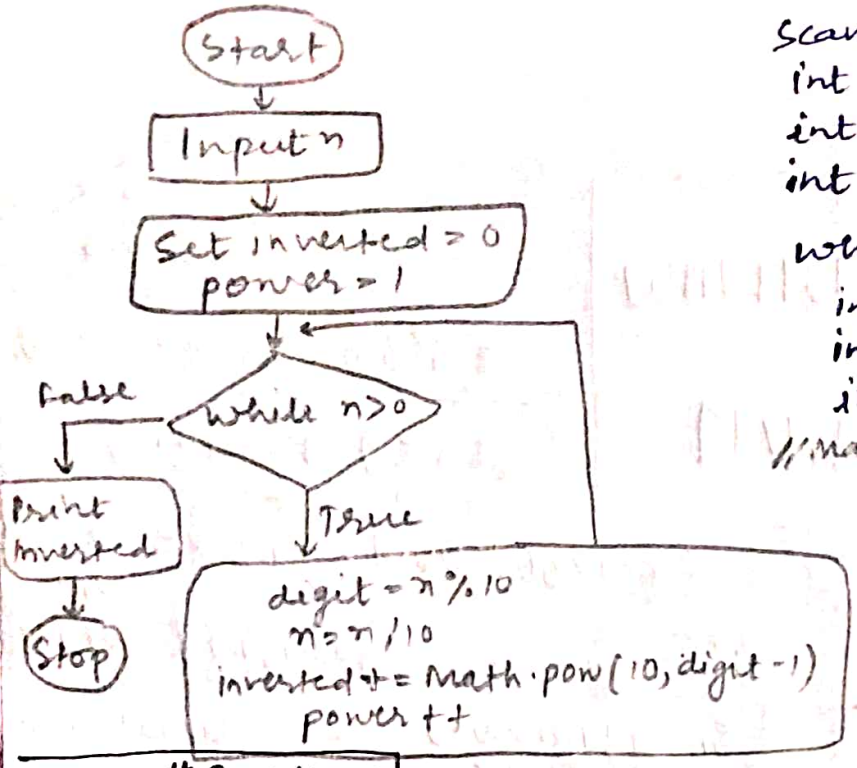
POSITION		DIGIT
①	⟶	$2 \times 10^0 = 2 \times 10^{1-1}$
②	⟶	$3 \times 10^1 = 3 \times 10^{2-1}$
③	⟶	$5 \times 10^2 = 5 \times 10^{3-1}$
④	⟶	$1 \times 10^3 = 1 \times 10^{4-1}$
⑤	⟶	$4 \times 10^4 = 4 \times 10^{5-1}$

```
int p = 1;
int inv = 0;
while (n > 0){
  int r = n % 10;
  int q = n / 10;
  n = q;
  // r @ p → p @ r → p * pow(10, r-1)
  inv = inv + p * (int)Math.pow(10, r-1);
  p++;
}
```

10	3 5 2 1 4	r	p
10	3 5 2 1	4 @ 1	
10	3 5 2	1 @ 2	
10	3 5	2 @ 3	
10	3	5 @ 4	
1	0	3 @ 5	

$4@1 = 1@4 = 1 \times 10^3 = 1000$
$1@2 = 2@1 = 2 \times 10^0 = 2$
$2@3 = 3@2 = 3 \times 10^1 = 30$
$5@4 = 4@5 = 4 \times 10^4 = 40000$
$3@5 = 5@3 = 5 \times 10^2 = 500$

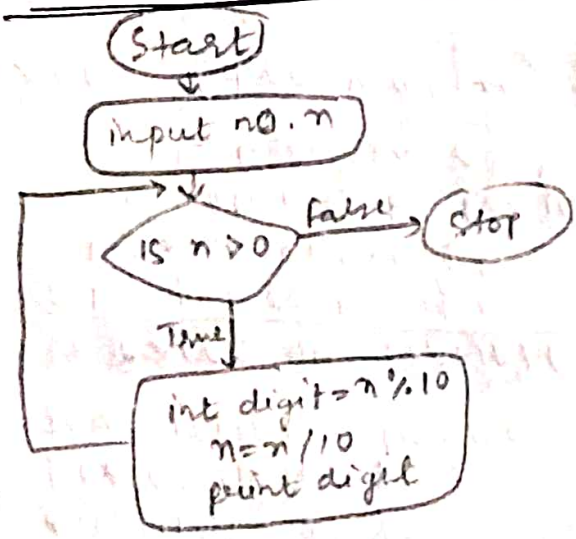
41532



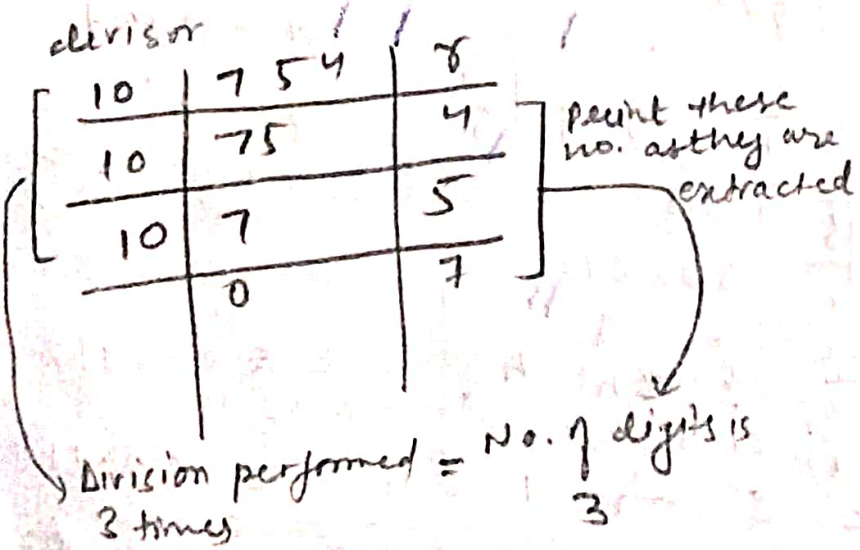

```
Scanner s = new Scanner (System.in);
int n = s.nextInt();
int inv=0; //inverted no. at start
int op = 1; //original place
while (n>0){
  int od=n%10; //last digit of Original no
  int id= op; //id inverted digit
  int ip = od; //ip inverted place
//make change to inverting ip and id
  inv= inv + (id*(int)Math.pow(10,ip-1));
   n=n/10;
   op++;
   }
   System.out.println(inv);
  }
}
```

Date: 6th Sept 2021
Lecture: 5

REVERSE A NO. (H.W)




```
public static void main (String [] args) {
Scanner s=new Scanner (System.in);
 int n = s.nextInt (); → 754
 while (n != 0)
 { int digit = n%10;
   n=n/10;
 System.out.println(digit);
   }
 }
}
```



Output

4
5
7

ROTATE A NO. (H.W)

I/P :→ 2 3 4 5 7

2

O/P :→ 5 7 2 3 4

→ 2 3 4 5 7

34572	-1	+1 → 72345
45723	-2	+2 → 57234
57234	-3	+3 → 45723
72345	-4	+4 → 34572
23457	-5	+5 → 23457

EX

2 7 3 5 | 1 6 —+2→ 162735

DIVISOR = 100
QUOTIENT = 2735
REMAINDER = 16
MULTIPLIER = 10000

100
10000

2735
+160000
162735

3 CASES TO HANDLE
- → +ve value of K
- → -ve value of K
- → value of K larger than no. of digits in n.

CASE 1 (+ve value of K)

K = 2

2 5 3 / 9 8 = n

n/100 = 98
REMAINDER = 98

QUOTIENT = 253

∴ K = 2
DIVISOR = 100
∴ 100 = 10^2 = 10^K
MULTIPLIER = 1000
1000 = 10^3 = $10^{(5-2)}$
= $10^{(No.\ of\ Digits - K)}$

AFTER DIVISION,
We,
Remainder × 1000 (98×1000)
+
Quotient (253)
Rotated No. = 98253
by +2

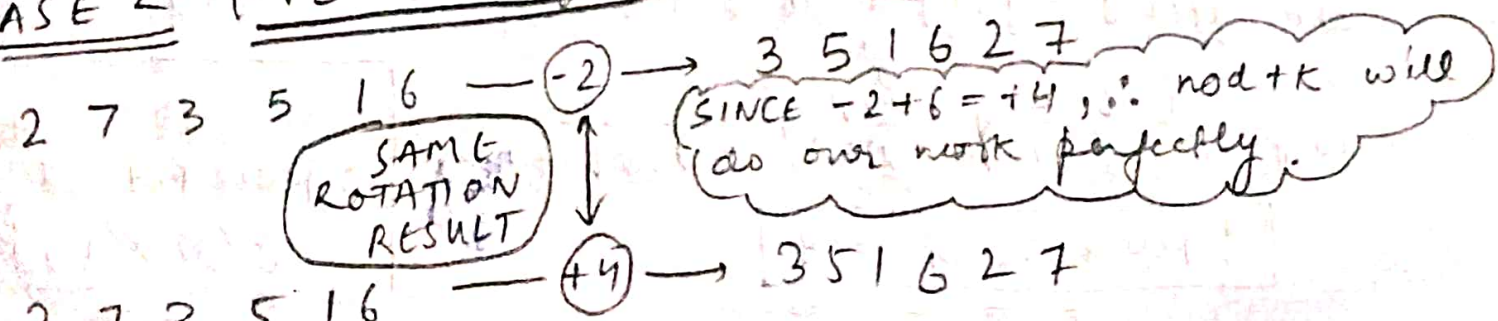
DIVISOR = $(10)^K$

MULTIPLIER = $(10)^{(NO.\ OF\ DIGITS - K)}$

Code (CASE 1)

```
Scanner sc = new Scanner(System.in);
int n = sc.nextInt(); ⟶ 25398
int k = sc.nextInt();
int temp = n; // temporarily storing no to perform divison
int nod = 0; // no. of digits
while (temp > 0) {
temp = temp / 10;
nod++;
} // 25398 will give nod = 5
int div = 1; // initialize divisor = 1
int mult = 1; // initialize multiplier = 1
 for(int i = 1; i <= nod; i++)
  {
   if(i <= k) // while the iterator is less than k
   { // we increase the value of divisor
    div = div * 10;
   }
   else { // else we increase the value of multiplier
   mult = mult * 10;
   }
  }
int q = n / div; // extracting the Quotient
int r = n % div; // Extracting the Remainder
int rot = r * mult + q; // Forming the rotated no.
System.out.println(rot);
}
}
```

CASE 2 (-ve value of k)



Code (Case 2)

```
if (k<0) {
   k= k+ nod;
}
```

Code (Case 3)

```
k = k % nod;
```

CASE 3 (value of K larger than no. of digits in n.)

25398 — same rotation

+5 → 25 3 98

-5 → 25 3 98

+10 → 25398

∴ Note:
If k multiple of nod
∴ rotation → no effect

Code (FINAL)

```
public static void main (String [] args)
{
Scanner s=new Scanner (System.in);
 int n = s.nextInt();
 int k = s.nextInt();

 int temp = n;
 int nod = 0;
 while (temp > 0) {
   temp = temp /10;
   nod++;
 }
 k = k % nod;  ⟶ // Case 3
 if (k < 0)    ⟶ // Case 2
 {  k = k + nod;
 }
 int div = 1;
 int mult = 1;
 for (int i=1; i<= nod; i++)
  { if (i <= k) {
      div = div * 10;
    }
    else {
     mult = mult * 10;
    }
 }
 int q = n / div;
 int r = n % div;
 int rot = r * mult + q;
 System.out.println (rot);
 }
```

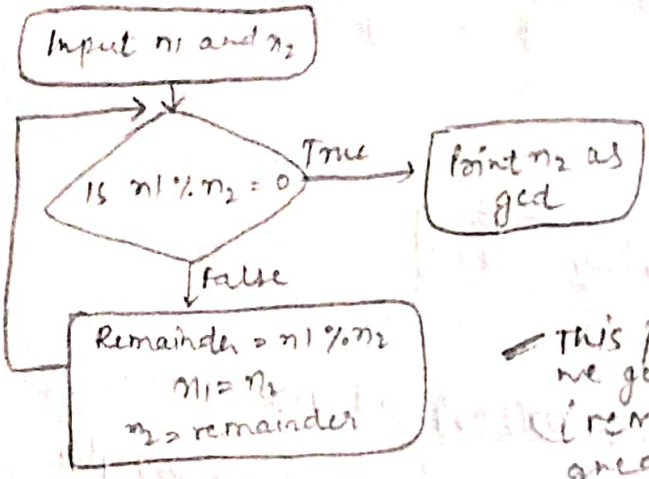
⟶ // Case 1

⑧ GCD and LCM

I/P: 36, 24 → O/P: 12, 72

TWO PART < GCD, LCM

∴ GCD Part



24 | 36 | 1
- 24 | 2
12 | 24
- 24
0

Remainder of previous division becomes divisor of next division

Previous Divisor becomes dividend in next division

→ This process of division is carried out until we get a divisor which divides evenly (i.e remainder 0) and that divisor is the greatest common divisor (GCD)

```
public static void main (String [] args) {
Scanner s = new Scanner (System.in);
int n1 = s.nextInt (); 36
int n2 = s.nextInt (); 24
while (n1 % n2 != 0)
{
  int rem = n1 % n2;
    n1 = n2;
    n2 = rem;
}
int gcd = n2; → 12
System.out.println (gcd);
}
```

```
int on1 = n1; → 36
int on2 = n2; → 24
```

```
int lcm = (on1 * on2)/gcd;
System.out.println (lcm);
```

$lcm = \frac{36 \times 24}{12} = 72$

∴ LCM Part

Relation b/w LCM and GCD

$$gcd * lcm = n1 * n2$$

$$\Downarrow$$

$$lcm = n1 * n2 / gcd$$

PRIME FACTORISATION OF A NUMBER

I/P: 1440 → O/P: 2 2 2 2 2 3 3 5

2	1440
2	720
2	360
2	180
2	90
3	45
3	15
5	5
	1

cannot divide further — take next larger integer

Prime Factors

2 2 2 2 2 3 3 5

Approaches

[2 to n] [2 to $\sqrt{n}$]

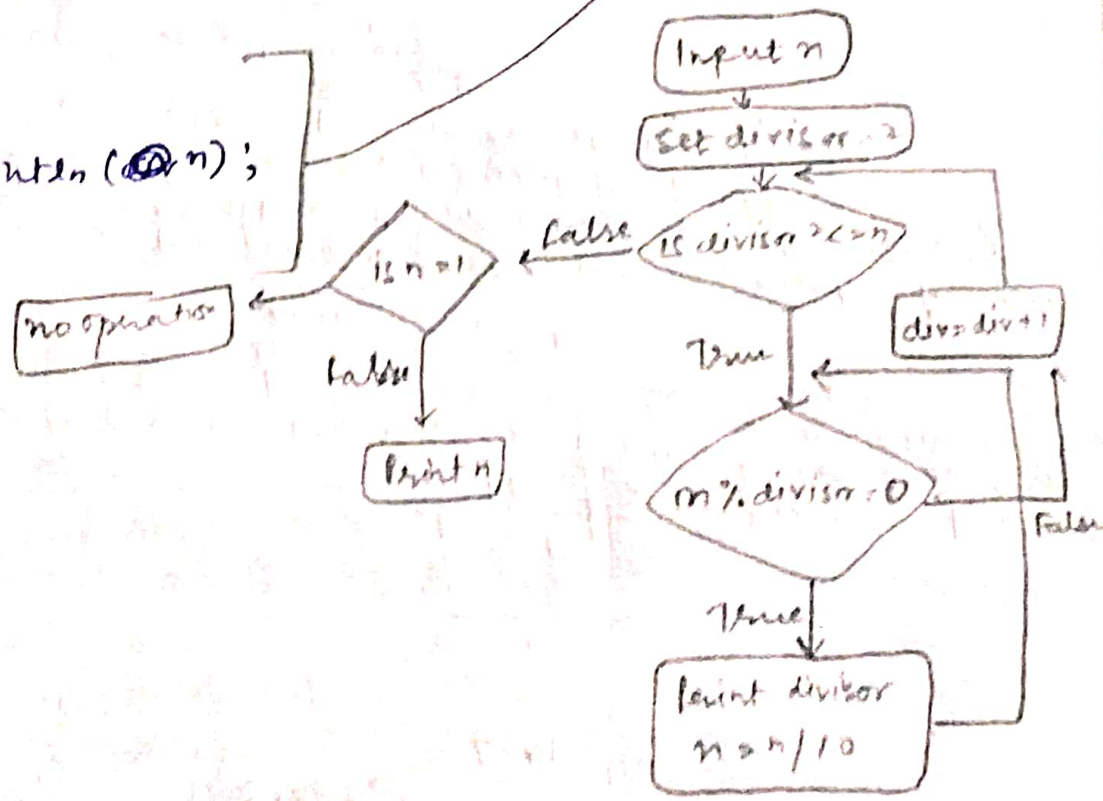
Best Optimized One

one drawback

some factors lies beyond the $\sqrt{n}$ range

eg: 2 | 45 | (3 3) → Print

```
public static void main (String [] args) {
Scanner s = new Scanner (System.in);
   int n = s.nextInt();
   for (int div=2 ; div * div <= n ; div++)
   {
     while (n%div == 0)
   {
    n=n/div;
    System.out.print (div + " ");
   }
   }
   if (n != 1)
   { System.out.println (n);
   }
}
```

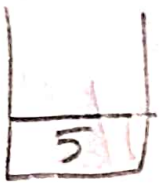


⑥ PYTHAGOREAN TRIPLET → (GEOMETRY IS KNOWLEDGE OF THE ETERNALLY EXISTENT.)

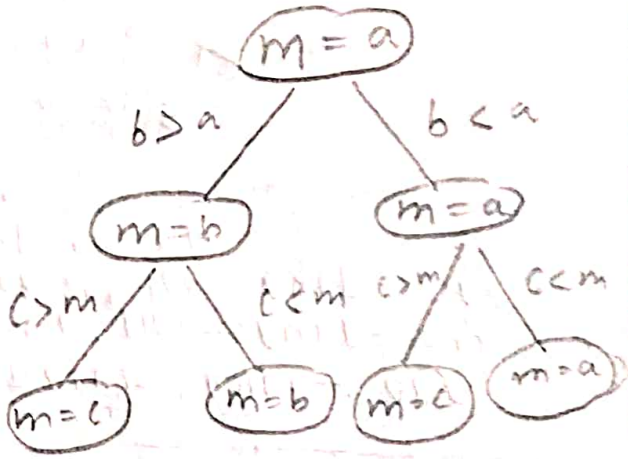
A pythagorean triplet consists of three positive integers a, b and c, such that $a^2 + b^2 = c^2$.
A triangle whose sides form a Pythagorean triple is called Pythagorean Triangle, and is necessarily of right Δ.

I/P: 5 3 4 → O/P: True

```
public static void main (String [] args){
Scanner sc = new Scanner (System.in);
  int a = sc.nextInt(); → 5
  int b = sc.nextInt(); → 3
  int c = sc.nextInt(); → 4
   int max = a;
   if (b >= max){
    max = b;
   }
   if (c >= max){
     max = c;
   }
   if (max == a){
   boolean flag = ((b*b + c*c) == (a*a));
   System.out.println (flag);
   }
else if (max == b){
   boolean flag = ((a*a + c*c) == (b*b));
   System.out.println (flag);
   }
   else {
   boolean flag = ((a*a + b*b) == (c*c));
   System.out.println (flag);
   }
}
```



SAUCHA JATHA STYLE



9 BENJAMIN BULBS

(★) → ON

1 ✓ (★)
2 ✓ ✓
3 ✓ ✓
4 ✓ ✓ ✓ (★)
5 ✓ ✓
6 ✓ ✓ ✓
7 ✓ ✓
8 ✓ ✓ ✓ ✓
9 ✓ ✓ ✓ (★)
10 ✓ ✓ ✓ ✓
11 ✓ ✓
12 ✓ ✓ ✓ ✓ ✓ ✓
13 ✓ ✓
14 ✓ ✓ ✓ ✓
15 ✓ ✓ ✓ ✓
16 ✓ ✓ ✓ ✓ ✓ (★)
17 ✓ ✓
18 ✓ ✓ ✓ ✓ ✓
19 ✓ ✓
20 ✓ ✓ ✓ ✓ ✓ ✓

$1 \rightarrow 1^2$
$4 \rightarrow 2^2$
$9 \rightarrow 3^2$
$16 \rightarrow 4^2$
→ PERFECT SQUARE'S

NUMBER
- EVEN NO. OF FACTORS → EVEN NO. OF CYCLES → OFF
- ODD NO. OF FACTORS → ODD NO. OF CYCLES → ON

∴ ALL PERFECT SQUARE'S HAVE PERFECT SQUARE'S

24 = 1×24, 2×12, 3×8, 4×6 / 24×1, 12×2, 8×3, 6×4 → (2n) → FACTORS

36 = 1×36, 2×18, 3×12, 4×9, 6×6 / 36×1, 18×2, 12×3, 9×4 → (2n+1) → FACTORS

INITIALLY → BULB → OFF
AFTER 1ST TICK → BULB → ON
AFTER 2ND TICK → BULB → OFF
AND SO ON.....

```
public static void main(String [] args){
 Scanner sc= new Scanner(System.in);
  int n= sc.nextInt(); → 100
 for (int i= 1; i*i<= n; i++){
   System.out.println(i*i);
   }
}
```

O/P : 1, 4, 9, 16, 25, 36, 49, 64, 81, 100

JAB QUES. HO JAYE TOH WHAMMAND NAHI KRNA KUKI EK QUES HAMESHA HOGA JO MERESE NAHI HOGA.

JAB QUES NA HO TOH BARNE KA NAHI KUKI EK DIN VO QUES HOGA

JAB QUES. HO JAYE TOH GHAMMAND NAHI KRNA
KYUKI EK QUES HAMESHA HOTA JO MERESE
NAHI HOTA.

JAB QUES NA HO TOH DARNE KA NAHI
KYUKI EK DIN VO QUES HOTA