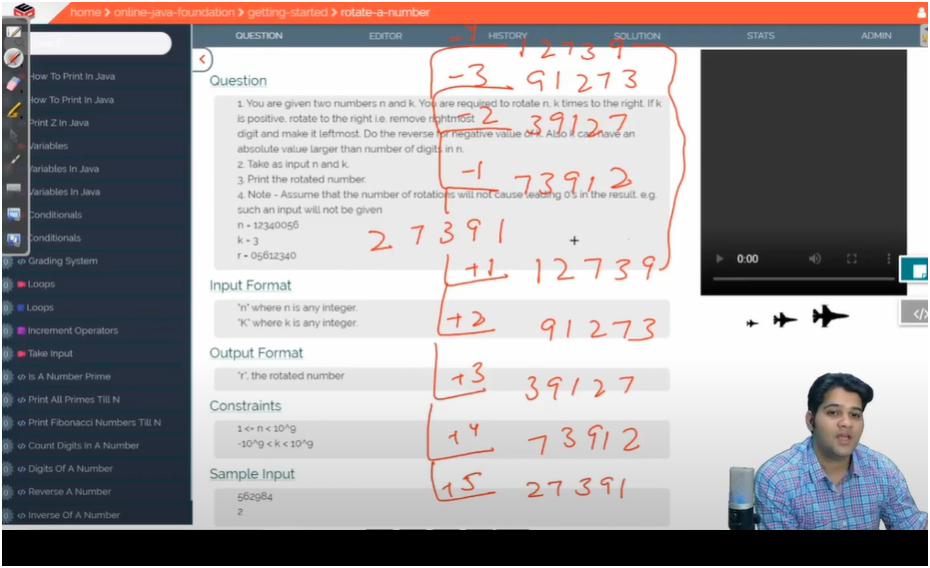
ROTATE A NUMBER



INPUT NUMBER:-127

POSITIVE

FIRST ROTATION 712

SECOND ROTATION 271

THIRD ROTATION 127

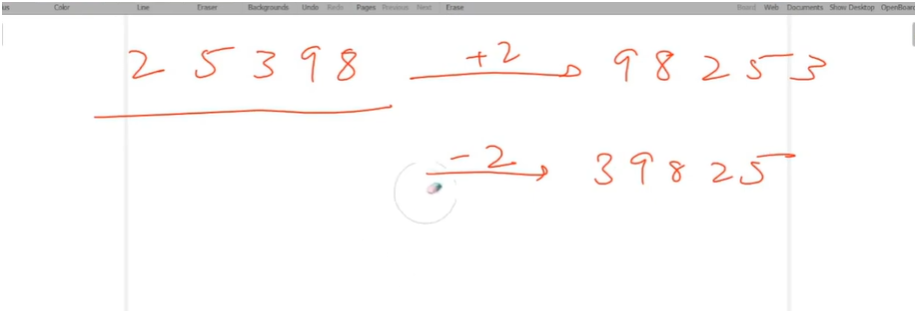
NEGATIVE

FIRST ROTATION 271

SECOND ROTATION 712

THIRD ROTATION 127

POSITIVE AND NEGATIVE ROTATION



FOR SHIFTING 2 POSITIVE POSTION

IF U MODULUS THE NUMBER BY 100 THEN WE GET LAST 2 DIGIT(REMAINDER)

AND IF U DIVIDE THE NUMBER BY 100 WE REMOVE THE LAST TWO DIGIT(QUOTIENT)

THEN MULTIPLY THE LAST 2 DIGIT BY 1000 MULTIPLIER AND THEN ADD IT WITH REMAINDER

DIVISOR LOGIC

25398/100=253

25398%100=98

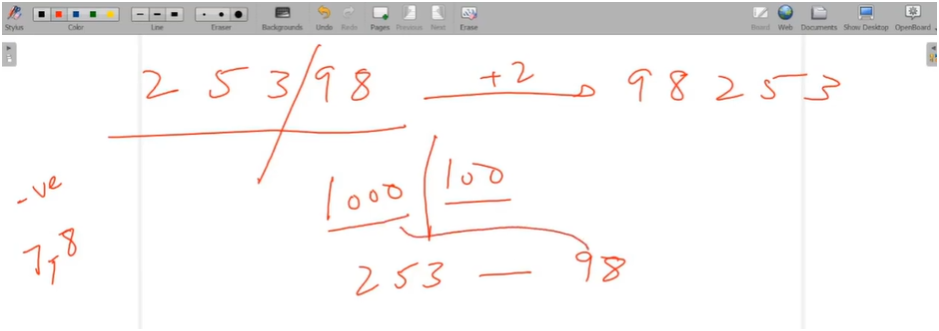
MULTIPLIER LOGIC

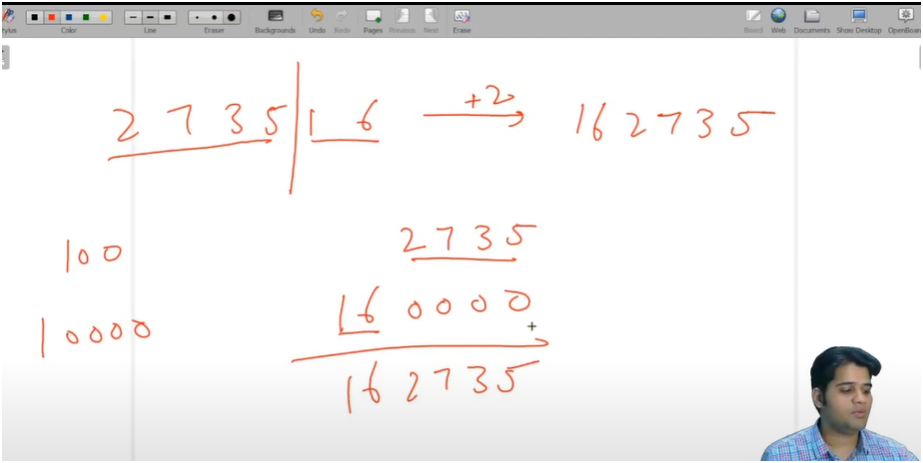
98\*1000=98000+253=98253

JITNA ROTATE KARNA AHE UTNA DIVISOR LO

AUR BAKHI KA UTNA MULTIPLIER LO

MULTIPLIER|DIVISOR



  
ALGO:-

====================== 🔄 ROTATE A NUMBER ALGORITHM ======================

🔢 Step 1: Take input number `n` and rotation count `k`.

🧮 Step 2: Count total digits (`nod`) in the number

- We store `n` in a temporary variable `temp`.

- Use a while loop to divide `temp` by 10 until it becomes 0.

- Increment `nod` on each step to get the total number of digits.

🧠 Step 3: Normalize the rotation count `k`

k = k % nod;

- This handles cases where rotation is greater than digit count.

E.g., if nod = 4, then rotating 6 times is same as rotating 6 % 4 = 2 times.

if (k < 0) {

k = k + nod;

}

- This handles negative rotation (left rotation).

E.g., rotating left by 1 (k = -1) is same as rotating right by nod - 1.

🔧 Step 4: Compute `div` and `mult`

- `div` is used to separate last `k` digits → n % div

- `mult` is used to shift the remaining digits → r = q + mult \* remainder

For example, if k = 2 and nod = 5, then:

div = 10^k = 100

mult = 10^(nod - k) = 1000

🔁 Step 5: Divide the number

- q = n / div → first part (left side of rotation)

- r = n % div → last part (right side to rotate front)

🔚 Step 6: Compute final rotated number

- r = r \* mult + q

🖨️ Step 7: Print the result