**Detailed Revision Notes on While Loops**

**Introduction to While Loops**

The while loop in programming allows us to execute a block of code repeatedly as long as a specified condition remains true. This is particularly useful for tasks that require repetitive actions, such as printing a message multiple times or processing items in a list.

**Basic Syntax of While Loops**

The syntax for while loops generally follows this structure:

initialization;

while (condition) {

// code to be executed

updation;

}

**Example: Printing "Hello" Multiple Times**

To understand the concept, let’s consider a simple example where we need to print "Hello" five times. We could do it manually as shown:

System.out.print("Hello");

System.out.print("Hello");

System.out.print("Hello");

System.out.print("Hello");

System.out.print("Hello");

However, this approach becomes impractical for large numbers of repetitions. Instead, we use a while loop:

int count = 0;

while (count < 5) {

System.out.print("Hello");

count++;

}

**Detailed Steps in a While Loop**

1. **Initialization**: Set up any variables that you'll need within the loop.
2. **Condition**: The loop continues to execute as long as this condition is true.
3. **Repetitive Task**: The code that you want to repeat.
4. **Updation**: Modify the variables used in the condition to eventually break the loop.

**Illustration with Example**

Consider the following example where we print the first 10 natural numbers.

int i = 1;

while (i <= 10) {

System.out.print(i + " ");

i++;

}

**Key Concepts Explored in Class**

**Iteration**

Iteration refers to the number of times the loop executes. For example:

int i = 1;

while (i <= 4) {

System.out.print(i + " ");

i++;

}

This loop iterates four times, printing numbers from 1 to 4.

**Managing Loop Conditions**

If not handled carefully, loops can become infinite or cause a "Time Limit Exceeded" error. For instance:

int i = 0;

while (i <= 10) {

System.out.print(i + " ");

i = i \* i;

}

This will enter an infinite loop since i remains 0 indefinitely.

**Practical Problems Solved Using While Loops**

1. **Printing Multiples of a Number**: Calculate and print multiples of 4 until a given number N.
2. Scanner scn = new Scanner(System.in);
3. int N = scn.nextInt();
4. int count = 4;
5. while (count <= N) {
6. System.out.print(count + " ");
7. count += 4;
8. }
9. **Finding Perfect Squares**: Print all perfect squares up to a given number N.
10. Scanner scn = new Scanner(System.in);
11. int N = scn.nextInt();
12. int count = 1;
13. while (count \* count <= N) {
14. System.out.print(count \* count + " ");
15. count++;
16. }
17. **Reversing a Number**: Reverse the digits of a given integer N.
18. Scanner scn = new Scanner(System.in);
19. int N = scn.nextInt();
20. int reverse = 0;
21. while (N > 0) {
22. int digit = N % 10;
23. reverse = reverse \* 10 + digit;
24. N /= 10;
25. }
26. System.out.print(reverse);

**Important Analogies and Notes from the Class**

* **Analogies**: Iterations in a loop are often likened to steps on a staircase — each step brings you closer to the end of the task.
* **Edge Cases**: Handle special scenarios such as zero or negative inputs carefully to avoid infinite loops or logic errors.

**Quizzes and Solutions Discussed**

1. **Quiz Example**: Predicting the output of a given segment of code.
2. int i = 1;
3. while (i <= 10) {
4. i = i \* i;
5. System.out.print(i + " ");
6. i++;
7. }
   * **Choices**: The correct answer is 1 4 25 since the iterations proceed as described.
8. **Update and Condition Management**: Add digits to the back of a number.
9. Let r = 13 & d = 4
10. We want to add d behind r i.e we need number 134
11. So r \* 10 = 130
12. r \* 10 + d => 130 + 4 = 134

**Summary**

Using while loops effectively requires understanding initialization, condition checking, repetitive tasks, and updates within the loop. These concepts enable us to solve various computational problems more efficiently than manual repetition.

<https://github.com/KingsGambitLab/Lecture_Notes/blob/non-dsa/Academy%20DSA%20Typed%20Notes/Java%20Refresher/Refresher%20While%20Loop.md>

DRY RUN:

Manually working through the code to emulate its execution.

WHILE LOOP:

A control flow statement that repeats a block of code as long as a specified condition is true.

INITIALIZATION IN WHILE LOOP:

Setting a variable to a starting value before entering the while loop.

CONDITIONS IN WHILE LOOP:

The logical test that determines whether the while loop will execute.

UPDATION IN WHILE LOOP:

Modifying the variable’s value to ensure the loop progresses towards termination.

PRINT FROM 1 TO N:

Use a counter starting from 1 and increment it until it reaches N.

PRINT FROM N TO 1:

Use a counter starting from N and decrement it until it reaches 1.

PRINT ODD NUMBERS FROM 1 TO N:

Using a while loop and checking if a number is odd to print it.

PRINT EVEN NUMBERS FROM N TO 1:

Using a while loop and checking if a number is even to print it in reverse order.

SUM OF DIGITS:

Summing up each digit of a number by extracting digits using modulo and integer division.

EXTRACT LAST DIGIT:

Using the modulo operation to extract the last digit of a number.

REMOVE LAST DISIT:

Using integer division by 10 to remove the last digit of a number.

INFINITE LOOP:

Occurs if the loop’s exit condition is never met, causing it to run indefinitely.

PERFECT SQUARE:

An integer that is the square of another integer.

<https://github.com/KingsGambitLab/Lecture_Notes/blob/non-dsa/Academy%20DSA%20Typed%20Notes/Java%20Refresher/Refresher%20While%20Loop.md>

POST INCREMENT:

Increases the value of a variable after its current value is used in an expression.

PRE INCREMENT:

Increases the value of a variable before its current value is used in an expression.

WHILE LOOP:

A control flow statement that allows repetitive execution as long as a given condition is true.

FOR LOOP:

A looping structure that iterates based on an initial condition, a termination condition, and an increment step.

CONTINUE STATEMENT:

Skips the current iteration of a loop and proceeds with the next iteration.

BREAK STATEMENT:

Exists the nearest enclosing loop prematurely, skipping the remaining code in the loop.

SCOPE OF VARIABLE:

Defines the portion of the code where the variables is accessible.

INITIALIZATION:

The process of assigning an initial value to a variable in a loop.

CONDITION:

A Boolean expression controlling the flow of loop iterations.

LOOP BODY:

The main code block that executes in each iteration of a loop.

COUNT VARIABLE:

A variable often used to control the number of iterations in looping structures.

TEST CASES:

Predefined inputs used to test the correctness of a programming problem solution.

WHILE LOOP:

Initialization;

while (Condition) {

//Operation

Increment/Decrement;

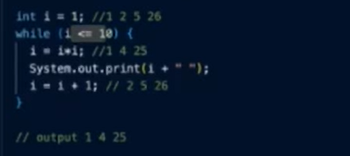
}

On chat when we given code to someone give in below format

```

YOUR CODE HERE

```



infinite

public class Main {

  public static void main(String[] args) {

   // print from 1 to N

   Scanner ip = new Scanner(System.in);

   int n=ip.nextInt();

   int i=1;

   while(i<=n){

    System.out.println(i);

    i++;

   }

   // Print N to 1

   System.out.println("---------");

   int n2=ip.nextInt();

   while(n2>=1){

    System.out.println(n2);

    n2--;

   }

    System.out.println("---------");

   // print odd values from 1 to n

   int n3=ip.nextInt();

   int i3=1;

   while(i3<=n3){

    if(i3%2!=0){

      System.out.println(i3);

    }

    i3++;

   }

  System.out.println("---------");

   // print odd values from 1 to n; below is better as its take less number of itersation

   int n4=ip.nextInt();

   int i4=1;

   while(i4<=n4){

      System.out.println(i4);

    i4+=2;

   }

   System.out.println("---------");

   // print even values from 1 to n

   int n5=ip.nextInt();

   int i5=2;

   while(i5<=n5){

      System.out.println(i5);

    i5+=2;

   }

    System.out.println("---------");

    // print multiple of 4 till n

   int n6=ip.nextInt();

   int i6=4;

   while(i6<=n6){

      System.out.println(i6);

    i6+=4;

   }

   System.out.println("---------");

    // print square of number till n

   int n7=ip.nextInt();

   int i7=1;

   while(i7\*i7<=n7){

      System.out.println(i7\*i7);

    i7++;

   }

  }

}

CUSTOM INPUT:

3

3

5

5

5

16

30

OUTPUT:

1

2

3

---------

3

2

1

---------

1

3

5

---------

1

3

5

---------

2

4

---------

4

8

12

16

---------

1

4

9

16

25

Given a number, print its digit.

e.g., 6531 then 1 3 5 6

public class Main {

  public static void main(String[] args) {

    int num=new Scanner(System.in).nextInt();

    while(num>0){

      System.out.println(num%10);

      num=num/10;

    }

  }

}

CUSTOM INPUT:

6531

OUTPUT:

1

3

5

6

Given a integer n we have to print sum of its digit

public class Main {

  public static void main(String[] args) {

    int num=new Scanner(System.in).nextInt();

    int sum=0;

    while(num>0){

      sum+=num%10;

      num=num/10;

    }

    System.out.print(sum);

  }

}

CUSTOM INPUT:  
6531

OUTPUT:

15

Given a number reverse it

WITH STRING DATA TYPE:

public class Main {

  public static void main(String[] args) {

    int num=new Scanner(System.in).nextInt();

    String rev="";

    while(num>0){

      rev+=num%10;

      num=num/10;

    }

    System.out.print(rev);

  }

}

WITH INT DATA TYPE:

public class Main {

  public static void main(String[] args) {

    int num=new Scanner(System.in).nextInt();

    int rev=0;

    while(num>0){

      rev=rev\*10+num%10;

      num=num/10;

    }

    System.out.print(rev);

  }

}

CUSTOM INPUT:

6531

OUTPUT:

1356

DRY RUN FOR INT DATATYPE:

num=6531 rev=0

rev=rev\*10+num%10;

rev=0\*10+6531%10 🡪 0+1🡪1 rev=1

num=num/10

num=6531/10 🡪653 num=653

rev=rev\*10+num%10;

rev=1\*10+653%10 🡪 10+3🡪13 rev=13

num=num/10

num=653/10🡪65 num=65

rev=rev\*10+num%10;

rev=13\*10+65%10 🡪 130+5🡪135 rev=135

num=num/10

num=65/10🡪6 num=6

rev=rev\*10+num%10;

rev=135\*10+6%10 🡪 1350+6🡪1356 rev=1356

**Q1. From top to down**

**Problem Description**

Write a program that takes a positive integer **N** as input from the user and prints all natural numbers **from 1 to N**, with each number **followed by a space** (including the last number).  
**Problem Constraints**

1 <= N <= 1000000  
**Input Format**

A single line representing **N**  
**Output Format**

**N** space separated integers from 1 to N, with each number **followed by a space**, including the last number.  
**Example Input**

Input 1:

5

Input 2:

10  
**Example Output**

Output 1:

1 2 3 4 5

Output 2:

1 2 3 4 5 6 7 8 9 10

CODE:

public class Main {

    public static void main(String[] args) {

        int num=new Scanner(System.in).nextInt(),i=1;

        while(i<=num) System.out.print(i+++" ");

    }

}

CODE2:

public class Main {

    public static void main(String[] args) {

        long N=new Scanner(System.in).nextLong();

        long i=1;

        if(N>0){

            while(i<=N){

                System.out.print(i+" ");

                i++;

            }

        }

    }

}

**Q2. From down to top**

**Problem Description**

Write a program to print all Natural numbers from **N** to **1** where you have to take N as input from user  
**Problem Constraints**

1 <= N <= 10000000  
**Input Format**

A single line representing **N**  
**Output Format**

**N** space separated integers from N to 1.  
**Example Input**

Input 1:

5

Input 2:

10  
**Example Output**

Output 1:

5 4 3 2 1

Output 2:

10 9 8 7 6 5 4 3 2 1

CODE:

public class Main {

    public static void main(String[] args) {

        int num=new Scanner(System.in).nextInt();

        while(num>0) System.out.print(num--+" ");

    }

}

CODE2:

public class Main {

    public static void main(String[] args) {

        Scanner ip=new Scanner(System.in);

        long N=ip.nextLong();

        if(N==0) System.out.print(N);

        else{

            while(N>0){

                System.out.print(N+" ");

                N--;

            }

        }

    }

}

**Q3. Odd Game**

**Problem Description**

Write a program to print all **odd numbers** from**1 to N** where you have to take N as input from user. Here N is inclusive.  
**Note:**  
Each number should be **followed by a space**, including the last number.  
**Problem Constraints**

1 <= N <= 2000000  
**Input Format**

A single line representing **N**  
**Output Format**

All **odd numbers** from 1 to N, each number **followed by a space**, including the last number.  
**Example Input**

Input 1:

5

Input 2:

10  
**Example Output**

Output 1:

1 3 5

Output 2:

1 3 5 7 9

CODE:

public class Main {

    public static void main(String[] args) {

        int num=new Scanner(System.in).nextInt();

        int i=1;

        while(i<=num){

            System.out.print(i+" ");

            i+=2;

        }

    }

}

**Q4. Even Game**

**Problem Description**  
Write a program to print all even numbers from **1 to N** where you have to take **N** as input from the user.  
**Note:** Use while-loop OR for-loop, according to session flow.

**Note:** You can use int(input()) for obtaining user input of **number N**.  
**Problem Constraints**

1 <= N <= 1000000

**Input Format**

A single line representing N

**Output Format**

All even numbers from 1 to N are separated by spaces.

**Example Input**

Input 1:

5

Input 2:

10

**Example Output**

Output 1:

2 4

Output 2:

2 4 6 8 10

**Example Explanation**

Sample 1:

2 4

The even numbers between 1 to 5 are 2 and 4. So the numbers 2 and 4 are printed separated by space.

Output 2:

2 4 6 8 10

The even numbers between 1 to 10 are 2, 4, 6, 8 and 10. So the numbers 2, 4, 6, 8, 10 are printed separated by space.

CODE:

public class Main {

    public static void main(String[] args) {

        int num=new Scanner(System.in).nextInt();

        int i=2;

        while(i<=num){

            System.out.print(i+" ");

            i+=2;

        }

    }

}

**Q5. Summation Game**

**Problem Description**

Write a program to find sum all Natural numbers from **1** to **N** where you have to take N as input from user  
**Problem Constraints**

1 <= N <= 1000  
**Input Format**

A single line representing **N**  
**Output Format**

A single integer showing sum of all Natural numbers from **1 to N**  
**Example Input**

Input 1:

5

Input 2:

10  
**Example Output**

Output 1:

15

Output 2:

55

CODE:

public class Main {

    public static void main(String[] args) {

        int num=new Scanner(System.in).nextInt(),sum=0,i=1;

        while(i<=num) sum=sum+i++;

        System.out.println(sum);

    }

}

**Q6. Multiples of 4**

**Problem Description**  
Given an integer input **N**, print all multiples of **4** **less than** or **equal to** **N**.  
**Problem Constraints**

1 <= N <= 10000  
**Input Format**

Single line containing an integer N.  
**Output Format**

Space separated integers representing multiples of 4 less than or equal to N.  
**Example Input**

22  
**Example Output**

4 8 12 16 20  
**Example Explanation**

1 \* 4 = 4  
2 \* 4 = 8  
3 \* 4 = 12  
4 \* 4 = 16  
5 \* 4 = 20  
All are multiples of 4 less than 22

CODE:

public class Main {

    public static void main(String[] args) {

      int num=new Scanner(System.in).nextInt(),i=4;

      while(i<=num)

      {

          System.out.print(i+" ");

          i=i+4;

      }

    }

}

**Q7. Print perfect squares**

**Problem Description**

Take an input of a number **A**from the user. Print all perfect squares less than or equal to **A**.  
**Note** - Perfect squares are integers whose square root is an integer. (For Example: 16 is **perfect square** as √16 = 4, or 42 = 16)  
**Problem Constraints**

1 <= A <= 104  
**Input Format**

A single line consisting of a integer A.  
**Output Format**

Print perfect squares less than or equal to A in a single line in a space-separated manner.  
**Example Input**

Input 1:

20

Input 2:

100  
**Example Output**

Output 1:

1 4 9 16

Output 2:

1 4 9 16 25 36 49 64 81 100

CODE:

public class Main {

    public static void main(String[] args) {

        int num=new Scanner(System.in).nextInt(),i=1;

        while(i\*i<=num){

            System.out.print(i\*i+" ");

            i++;

        }

    }

}

**Q8. First vs Last**

**Problem Description**

Write a program that asks the user to input a **number T,** indicating the number of test cases.  
Then, for each test case, ask for input a **number N** and print the **first** and **last** digits of **N**.  
**Problem Constraints**

1 <= T <= 1000

0 <= N <= 100000000  
**Input Format**

First line is **T** which means number of test cases.

Each next **T** lines contain an integer **N**.  
**Output Format**

**T** lines each containing two space separated integers representing first and last digits of the input integer.  
**Example Input**

Input 1:

2

5

1001

Input 2:

2

10023

1589  
**Example Output**

Output 1:

5 5

1 1

Output 2:

1 3

1 9  
**Example Explanation**

Explanation 1:

5 has fist and last digits same.

1001 also has fist and last digits same.

Explanation 2:

10023 has 1 as first digit and 3 as last digit.

1589 has 1 as first digit and 9 as last digit.

CODE:

public class Main {

    public static void main(String[] args) {

        Scanner ip=new Scanner(System.in);

        int numOfInt=ip.nextInt();

        while(numOfInt>0){

            int num=ip.nextInt();

            int lastDigit=num%10;

            int firstDigit=0;

            while(num>0){

                firstDigit=num%10;

                num=num/10;

            }

            System.out.println(firstDigit+" "+lastDigit);

            numOfInt--;

        }

    }

}

CODE2:

public class Main {

    public static void main(String[] args) {

        Scanner ip=new Scanner(System.in);

        int T=ip.nextInt();

        while(T>=1){

            long N=ip.nextLong();

            if(N<10)System.out.println(N+" "+N);

            else{

            long last=N%10;

            while(N>=10){

                N=N/10;

            }

            System.out.print(N+" ");

            System.out.println(last);

            }

            T--;

        }

    }

}

**Q9. Count the digits**

**Problem Description**  
Take T (number of test cases) as input.  
For each test case, take integer N as input and Print the count of digits of that number.

Note: **No of digits for number 0 is considered as 1.**

**Problem Constraints**

1 <= T <= 1000  
0 <= N <= 100000000

**Input Format**

The first line is the number T which denotes the total number of test cases.

Next T lines contain an integer N for which you have to print the number of digits.

**Output Format**

For T different Numbers, Print the number of digits in separate lines.

**Example Input**

Input 1:

2

0

1

Input 2:

2

100

10101

**Example Output**

Output 1:

1

1

Output 2:

3

5

**Example Explanation**

Explanation 1:

0 and 1 both have only one digit.

Explanation 2:

100 has three digits and 10101 has 5 digits.

CODE:

public class Main {

    public static void main(String[] args) {

        Scanner ip=new Scanner(System.in);

        int numOfInt=ip.nextInt();

        while(numOfInt>0){

            int num=ip.nextInt();

            if(num==0){

                System.out.println(1);

            }else{

                int numOfDigitCount=0;

                while(num>0){

                    num=num/10;

                    numOfDigitCount+=1;

                }

                System.out.println(numOfDigitCount);

            }

            numOfInt--;

        }

    }

}

CODE:

public class Main {

    public static void main(String[] args) {

        Scanner ip=new Scanner(System.in);

        short T=ip.nextShort();

        while(T>0){

            long N=ip.nextLong();

            if(N<10)System.out.println("1");

            else{

                short count=0;

            while(N>0){

                count++;

                N=N/10;

            }

            System.out.println(count);

            }

            T--;

        }

    }

}

**Q10. Sum the digits**

**Problem Description**  
You take a number of test cases, denoted by **T as input**.For each test case, you should take integers **N** as input. Your task is to calculate and print the sum of the digits of the given number N.

**Problem Constraints**

1 <= T <= 1000

0 <= N <= 100000000

**Input Format**

The first line is T which means the total number of test cases.

Each of the next T lines contain an integer N.

. **Output Format**

**T** lines each containing one integer representing the sum of the digits of the input integer.

**Example Input**

Input 1:

2

5

1001

Input 2:

2

123

1589

**Example Output**

Output 1:

5

2

Output 2:

6

23

**Example Explanation**

Explanation 1:

5 has only 1 digit hence sum is 5.

For the number 1001, the digits are 1,0,0,1. The Sum(1001) = 1+0+0+1 = 2.

Explanation 2:

For the number 123, the digits are 1,2,3. The Sum(123) = 1+2+3 = 6.

For the number 1589, the digits are 1,5,8,9. The Sum(1589) = 1+5+8+9 = 23.

CODE:

public class Main {

    public static void main(String[] args) {

        Scanner ip=new Scanner(System.in);

        int numOfint=ip.nextInt();

        while(numOfint>0){

            int num=ip.nextInt();

            if(num<10){

                System.out.println(num);

            }else{

                int sum=0;

                while(num>0){

                    sum=sum+num%10;

                    num=num/10;

                }

                System.out.println(sum);

            }

            numOfint--;

        }

    }

}

**Q1. Sum of Odds – Easy**

**Problem Description**

Take an integer **A** as input. You have to print the sum of all odd numbers in the range **[1, A]**.  
**Problem Constraints**

1 <= **A** <= 1000  
**Input Format**

First and only line contains a single positive integer **A**.  
**Output Format**

Print the required sum in a single line.  
**Example Input**

Input 1:

1

Input 2:

4   
**Example Output**

Output 1:

1

Output 2:

4   
**Example Explanation**

Explanation 1:

For A = 1, **1** is the only odd number which lies in the range **[1, 1]**.

Explanation 2:

For A = 4, Odd numbers **1** and **3** lie in the range **[1, 4]**.

Sum = 1 + 3 = 4.

CODE:

public class Main {

    public static void main(String[] args) {

        int num=new Scanner(System.in).nextInt(),sum=0;

        while(num>=0){

            if(num%2!=0)sum=sum+num;

            num--;

        }

        System.out.print(sum);

    }

}

CODE2:

public class Main {

    public static void main(String[] args) {

        Scanner ip = new Scanner(System.in);

        int A=ip.nextInt();

        int i=1,sum=0;

        while(i<=A){

            sum=sum+i;

            i+=2;

        }

        System.out.print(sum);

    }

}

**Q2. Sum of Evens – easy**

**Problem Description**

You are given a positive integer **A**. You have to print the sum of all even numbers in the range **[1, A]**.  
**Problem Constraints**

1 <= **A** <= 1000  
**Input Format**

First and only line contains a single positive integer **A**.  
**Output Format**

Print the required sum in a single line.  
**Example Input**

Input 1:

1

Input 2:

4   
**Example Output**

Output 1:

0

Output 2:

6   
**Example Explanation**

Explanation 1:

For A = 1, there are no even number which lies in the range **[1, 1]**.

Explanation 2:

For A = 4, Even numbers **2** and **4** lie in the range **[1, 4]**.

Sum = 2 + 4 = 6.

CODE:

public class Main {

    public static void main(String[] args) {

        int num=new Scanner(System.in).nextInt(),sum=0;

        while(num>0){

            if(num%2==0) sum=sum+num;

            num--;

        }

        System.out.println(sum);

    }

}

CODE2:

public class Main {

    public static void main(String[] args) {

        int sum=0,i=2;

        Scanner ip=new Scanner(System.in);

        int A=ip.nextInt();

        while(i<=A){

            sum=sum+i;

            i+=2;

        }

        System.out.print(sum);

    }

}

**Q3. Easy Power**

**Problem Description**

You are given two integers **A** and **B**. You have to find the value of **AB**.

**NOTE:** The value of **AB** will always be less than or equal to **109**.  
**Problem Constraints**

1 <= **A, B**<= 1000  
**Input Format**

First line of the input contains a single integer **A**.

Second line of the input contains a single integer **B**.  
**Output Format**

Print a single integer in single line.  
**Example Input**

Input 1:

2

3

Input 2:

1

10   
**Example Output**

Output 1:

8

Output 2:

1   
**Example Explanation**

Explanation 1:

For A = 2 and B = 3, the value of 23 = 2 \* 2 \* 2 = 8.

Explanation 2:

For A = 1 and B = 10, the value of 110 = 1.

CODE:

public class Main {

    public static void main(String[] args) {

         Scanner ip=new Scanner(System.in);

         int num=ip.nextInt(),pow=ip.nextInt(),result=1;

         while(pow>0){

             result\*=num;

             pow--;

         }

         System.out.print(result);

    }

}

**Q4. Multiplication Table!**

**Problem Description**

Take a number **A** as input, print its multiplication table having the first 10 multiples.  
**Problem Constraints**

1 <= **A** <= 1000  
**Input Format**

First line contains a single integer **A**.  
**Output Format**

Print **10** lines, ith line containing ith multiple.  
**Example Input**

Input 1:

2

Input 2:

3   
**Example Output**

Output 1:

2 \* 1 = 2

2 \* 2 = 4

2 \* 3 = 6

2 \* 4 = 8

2 \* 5 = 10

2 \* 6 = 12

2 \* 7 = 14

2 \* 8 = 16

2 \* 9 = 18

2 \* 10 = 20

Output 2:

3 \* 1 = 3

3 \* 2 = 6

3 \* 3 = 9

3 \* 4 = 12

3 \* 5 = 15

3 \* 6 = 18

3 \* 7 = 21

3 \* 8 = 24

3 \* 9 = 27

3 \* 10 = 30   
**Example Explanation**

Explanantion 1:

For A = 2, First 10 multiples of 2 are 2, 4, 6, 8, 10, 12, 14, 16, 18, 20

Explanation 2:

For A = 3, First 10 multiples of 3 are 3, 6, 9, 12, 15, 18, 21, 24, 27, 30

CODE:

public class Main {

    public static void main(String[] args) {

        int num=new Scanner(System.in).nextInt(),i=1;

        while(i<=10){

            System.out.println(num+" \* "+i+" = "+(num\*i));

            i++;

        }

    }

}

**Q5. Bank Account – 2**

**Problem Description**

You have been provided with a bank account that has an initial balance of **N** amount. You are now required to perform two operations on this account, namely, **ADD** and **SUBTRACT**.

* **ADD operation**: This operation increases the account balance by a certain amount and you are expected to print the updated balance after each **ADD** operation.
* **SUBTRACT operation**: This operation decreases the account balance by a certain amount and you are again expected to print the updated balance after each **SUBTRACT** operation.

However, if the amount you are trying to **subtract** (i.e., debit) from the account balance is greater than the current balance, you should print "**Insufficient Funds**" (without quotes) instead of the updated balance. In this case, the operation should be skipped, and the account balance should remain unchanged.

**Note :**Initial Amount N and Amount that is given are larger numbers.  
**Problem Constraints**

1 <= N, X <= 1011

1 <= Number of operations <= 105  
**Input Format**

* The **initial balance** in the bank account is provided as a single **integer N**.
* The **number of operations** to be performed on the bank account is provided as a single **integer M**.
* Each of the **next M** lines contains two space-separated integers **Type and Amount(X)**.
* The value of Type can either be 1 or 2. If **Type is 1**, then the **ADD** operation needs to be performed, and if **Type is 2**, then the **SUBTRACT** operation needs to be performed.
* The value of **Amount(X)** represents the amount of money to be added or subtracted from the account.

**Output Format**

Print Amount in the bank balance after each operation on a new line.  
**Example Input**

1000

3

1 500

2 1400

2 500  
**Example Output**

1500

100

Insufficient Funds  
**Example Explanation**

Initially bank balance is 1000.

First operation, ADD 500, bank balance becomes 1500, print it.

Second operation, SUBTRACT 1400, bank balance becomes 100, print it.

Third operation, SUBTRACT 500, print "Insufficient Funds".

CODE:

public class Main {

    public static void main(String[] args) {

        Scanner ip=new Scanner(System.in);

        long amount=ip.nextLong(),numOfOperations=ip.nextInt();

        while(numOfOperations>0){

            int operation=ip.nextInt();

            if(operation==1){

                long amountToBeAdd=ip.nextLong();

                amount=amount+amountToBeAdd;

                System.out.println(amount);

            }else if(operation==2){

                long amountToBeSubstract=ip.nextLong();

                if(amountToBeSubstract>amount){

                    System.out.println("Insufficient Funds");

                }else{

                    amount=amount-amountToBeSubstract;

                    System.out.println(amount);

                }

            }

            numOfOperations--;

        }

    }

}

CODE2:

public class Main {

    public static void main(String[] args) {

        // YOUR CODE GOES HERE

        // Please take input and print output to standard input/output (stdin/stdout)

        // DO NOT USE ARGUMENTS FOR INPUTS

        // E.g. 'Scanner' for input & 'System.out' for output

        Scanner ip=new Scanner(System.in);

        long N=ip.nextLong();

        int M=ip.nextInt();

        while(M>0){

            short type=ip.nextShort();

            long X=ip.nextLong();

            if(type==1) {

                N=N+X;

                System.out.println(N);

            }

            else {

                if(X>N)System.out.println("Insufficient Funds");

                else {

                    N=N-X;

                     System.out.println(N);

                }

            }

            M--;

        }

    }

}

**Q6. Palindromic Integer**

**Problem Description**

You are given an**integer A** as input and you need to determine whether it is a **palindrome** or **not**.  
A **palindrome integer** is one whose digits, when reversed, result in the same number.  
For **example**, 121 is a palindrome because its reverse is also 121, but 123 is not a palindrome because its reverse is 321.  
**Note:**The given integer will not have any leading zeros.  
**Problem Constraints**

1 <= A <= 106  
**Input Format**

First and the only line contains a single integer **A**.  
**Output Format**

Print **Yes** if it is palindromic, else print **No**.  
**Example Input**

Input 1:

120

Input 2:

1001

Input 3:

131   
**Example Output**

Output 1:

No

Output 2:

Yes

Output 3:

Yes   
**Example Explanation**

Explanation 1:

For A = 120, reverse(A) = reverse(120) = 021 = 21 (removing leading zeroes). 120 is not equal to 21

Explanation 2:

For A = 1001, reverse(A) = reverse(1001) = 1001, which is same as A.

Explanation 3:

For A = 131, reverse(A) = reverse(131) = 131, which is same as A.

CODE:

public class Main {

    public static void main(String[] args) {

        int num=new Scanner(System.in).nextInt();

        int originalNum=num,sum=0;

        while(num>0){

            sum=sum\*10+num%10;

            num=num/10;

        }

        System.out.println(originalNum==sum?"Yes":"No");

    }

}