

Learner Assignment Submission Format

Learner Details

- **Name:** H O Akash
 - **Enrollment Number:**
 - **Batch / Class:**
 - **Assignment: (Bridge Course Day 3)**
 - **Date of Submission:**
-

Problem Solving Activity 1.1

1. Program Statement

Repetitive Tasks List three tasks you perform regularly that involve repetition. For each:

1. What is being repeated?
 2. What determines when it stops?
-

2. Algorithm

List the step-by-step procedure your program will follow to solve the problem. Use numbered steps to describe the logic clearly.

3. Pseudocode

Write the pseudocode representing your algorithm in a structured format. It should resemble the actual code structure but use plain language or programming logic.

4. Program Code

Write your complete program code here. Use proper indentation, comments, and meaningful variable and function names.

Example (in Python)

```
def add(a, b):
```

return a + b

5. Test Cases

Present a table of test cases you used to validate your program. Include a mix of regular, boundary, and edge cases.

Test Case No.	Input	Expected Output	Actual Output	Status (Pass/Fail)
1				
2				
3				

6. Screenshots of Output

Paste or attach clear screenshots of your program's output. Ensure they match the test cases and show successful or failed results as applicable.

7. Observation / Reflection

Reflect on your experience while working on this assignment. Consider answering the following:

- What challenges did you face?
- What did you learn from completing this task?
- What would you improve or do differently next time?

Problem Solving Activity 3.2

1. Program Statement

Write how you would print “Hello!” 10 times without loops. Reflect on how loops make this easier for 1000 times.

2. Algorithm

1. Initialize a counter to 0.
 2. Set the desired number of iterations (e.g., 1000).
 3. Repeat the following steps until the counter reaches the desired number of iterations:
 - Print "Hello!".
 - Increment the counter by 1.
-

3. Pseudocode

BEGIN

SET iterations = 1000

SET counter = 0

WHILE counter < iterations

PRINT "Hello!"

counter = counter + 1

END WHILE

END

4. Program Code for 10 times without loop

```
import java.util.Scanner;

public class D3_2 {

    public static void main(String[] args) {

        System.out.println("Hello!");

        System.out.println("Hello!");
        System.out.println("Hello!");
        System.out.println("Hello!");
        System.out.println("Hello!");
        System.out.println("Hello!");
        System.out.println("Hello!");
        System.out.println("Hello!");
        System.out.println("Hello!");
        System.out.println("Hello!");

    }

}
```

4. Program Code for print 1000 times with loop

```
import java.util.Scanner;

public class D3_2 {

    public static void main(String[] args) {

        for (int i = 0; i < 1000; i++) {

            System.out.println("Hello!");

        }

    }

}
```

```
4e44029c04154844b0b411e
Hello!
Hello!
Hello!
Hello!
Hello!
Hello!
Hello!
Hello!
Hello!
Hello!
```

Repetition is a fundamental concept in programming: Loops allow us to repeat a set of instructions multiple times, making it easier to perform tasks that require repetition.

Problem Solving Activity 3.3

1. Program Statement

Countdown Print numbers from 10 to 1, using while loop then “Blastoff!”

2. Algorithm

1. Initialize a variable count to 10.
 2. Use a while loop to repeat the following steps until count is less than 1:
 - Print the current value of count.
 - Decrement count by 1.
 3. After the loop ends, print "Blastoff!"
-

3. Pseudocode

```
BEGIN
  SET count = 10
  WHILE count >= 1
    PRINT count
    count = count - 1
  END WHILE
  PRINT "Blastoff!"
END
```

4. Program Code

```
public class D3_3 {
    public static void main(String[] args) {
        int count = 10;
        while (count >= 1) {
```

```

        System.out.println(count);

        count--;

    }

    System.out.println("Blastoff!");

}

}

```

5. Test Cases

Present a table of test cases you used to validate your program. Include a mix of regular, boundary, and edge cases.

Test Case No.	Input	Expected Output	Actual Output	Status (Pass/Fail)
1	10,1,--	10 to 1 than blastoff	10 to 1 than blastoff	pass
2				
3				

6. Screenshots of Output


```
4e44029c04f54844b6b41
10
9
8
7
6
5
4
3
2
1
Blastoff!
PS C:\Users\akash\One
```

7. Observation / Reflection

- Loops are useful for repetitive tasks: The while loop in this program allows us to repeat the countdown sequence from 10 to 1 in a concise and efficient way.
- Control structures are essential for flow control: The while loop provides a way to control the flow of the program, allowing us to repeat a set of instructions until a certain condition is met.
- Simple programs can be powerful: This countdown program demonstrates how a simple program can be used to create an engaging and interactive experience.

Problem Solving Activity 3.4

1. Program Statement

Sum Until Zero Ask user for numbers repeatedly until they enter 0. Sum and print the total.

2. Algorithm

1. Initialize a variable sum to 0.
 2. Use a while loop to repeatedly ask the user for numbers:
 - Read the user's input and store it in num.
 - If num is 0, break out of the loop.
 - Otherwise, add num to sum.
 3. After the loop ends, print the total sum.
-

3. Pseudocode

BEGIN

SET sum = 0

WHILE TRUE

```
INPUT num
IF num == 0
    BREAK
END IF
sum = sum + num
END WHILE
PRINT "Total: " + sum
END
```

4. Program Code

```
import java.util.Scanner;

public class D3_4 {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        int sum = 0;

        while (true) {

            System.out.print("Enter a number (0 to stop): ");

            int num = scanner.nextInt();

            if (num == 0) {

                break;

            }

            sum += num;

        }

        scanner.close();

    }

}
```

```

        System.out.println("Total: " + sum);
    }
}

```

5. Test Cases

Present a table of test cases you used to validate your program. Include a mix of regular, boundary, and edge cases.

Test Case No.	Input	Expected Output	Actual Output	Status (Pass/Fail)
1	5,5,5, 5,4,4, 0	28	28	pass
2	10,10 ,0	20	20	pass
3	4,5,6, 7,8,0	30	30	pass

6. Screenshots of Output

```
Enter a number (0 to stop): 5
Enter a number (0 to stop): 5
Enter a number (0 to stop): 5
Enter a number (0 to stop): 5
Enter a number (0 to stop): 4
Enter a number (0 to stop): 4
Enter a number (0 to stop): 0
Total: 28
```

7. Observation / Reflection

- Loops are useful for handling user input: The while loop in this program allows us to repeatedly ask the user for numbers until they decide to stop.
- Conditional statements are essential for flow control: The if statement inside the loop provides a way to break out of the loop when the user enters 0.
- Programs can be designed to be user-friendly: By prompting the user to enter 0 to stop, we make it clear how to exit the program.

Problem Solving Activity 3.5

1. Program Statement

Guess the Number Generate a random number between 1 and 10. Ask user to guess. Provide feedback and loop until correct.

2. Algorithm

1. Generate a random number between 1 and 10.
2. Initialize a variable attempts to 0.
3. Use a while loop to repeatedly ask the user to guess:
 - Read the user's input and store it in user guess.
 - Increment attempts by 1.

- If user guess is less than the number to guess, print "Too low! Try again."
 - If user guess is greater than the number to guess, print "Too high! Try again."
 - If user guess is equal to the number to guess, print a congratulatory message with the number of attempts and break out of the loop.
-

3. Pseudocode

BEGIN

GENERATE random number between 1 and 10

SET attempts = 0

WHILE TRUE

 INPUT user guess

 attempts = attempts + 1

 IF user guess < number to guess

 PRINT "Too low! Try again."

 ELSE IF user guess > number to guess

 PRINT "Too high! Try again."

 ELSE

 PRINT "Congratulations! You guessed the number in " + attempts + " attempts."

 BREAK

 END IF

END WHILE

END

4. Program Code

```
import java.util.Random;
```

```
import java.util.Scanner;

public class D3_5 {

    public static void main(String[] args) {

        Random random = new Random();

        int numberToGuess = random.nextInt(10) + 1;

        Scanner scanner = new Scanner(System.in);

        int attempts = 0;

        while (true) {

            System.out.print("Guess a number between 1 and 10: ");

            int userGuess = scanner.nextInt();

            attempts++;

            if (userGuess < numberToGuess) {

                System.out.println("Too low! Try again.");

            } else if (userGuess > numberToGuess) {

                System.out.println("Too high! Try again.");

            } else {

                System.out.println("Congratulations! You guessed the
number in " + attempts + " attempts.");

                break;

            }

            scanner.close();

        }

    }

}
```

5. Test Cases

Present a table of test cases you used to validate your program. Include a mix of regular, boundary, and edge cases.

Test Case No.	Input	Expected Output	Actual Output	Status (Pass/Fail)
1	5	Too low! Try again	Too low! Try again	pass
2	10	Too high! Try again.	Too high! Try again.	pass
3	7	Congratulations! You guessed the number in 3 attempts.	Congratulations! You guessed the number in 3 attempts.	pass

6. Screenshots of Output

```

4e44029c04f54844b6b411e\redhat.java\jdt_ws\Day3_e95addc
Guess a number between 1 and 10: 4
Too low! Try again.
Guess a number between 1 and 10: 5
Too low! Try again.
Guess a number between 1 and 10: 6
Too low! Try again.
Guess a number between 1 and 10: 7
Congratulations! You guessed the number in 4 attempts.
PS C:\Users\akash\OneDrive\Desktop\StemupBridgecourse\D

```

7. Observation / Reflection

Random number generation adds an element of surprise: By generating a random number, we make the game more engaging and unpredictable.

Loops are useful for handling repeated user input: The while loop in this program allows us to repeatedly ask the user for guesses until they correctly guess the number.

Conditional statements provide feedback: The if-else statements inside the loop provide feedback to the user after each guess, helping them narrow down the range of possible numbers.

Problem Solving Activity 3.6

1. Program Statement

Infinite Loop Debugging Analyze

```
int counter = 0;
while (counter < 5) {
    System.out.println("Hello");
}
```

2. Solution

To fix the infinite loop, we need to update the value of counter inside the loop. One way to do this is to increment counter by 1 in each iteration

3. Solution Program Code

```
import java.util.Scanner;

public class D3_6 {

    public static void main(String[] args) {

        int counter = 0;

        while (counter < 5) {

            System.out.println("Hello");

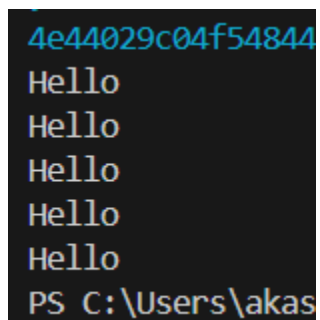
            counter++;

        }

    }

}
```

4. Screenshots of Output



```
4e44029c04f54844
Hello
Hello
Hello
Hello
Hello
PS C:\Users\akas
```

7. Observation / Reflection

The iteration in the loops are more important to fix the infinite loop, we need to update the value of counter inside the loop. One way to do this is to increment counter by 1 in each iteration

Problem Solving Activity 3.7

1. Program Statement

Even Numbers Print even numbers from 2 to 20 using a for loop.

2. Algorithm

1. Initialize a variable i to 2 (the first even number).
2. Use a for loop to repeat the following steps until i is greater than 20:
 - Print the current value of i.
 - Increment i by 2 (to move to the next even number).

3. Pseudocode

Begin

for i = 2 to 20 step 2

 printi

End for

end

4. Program Code

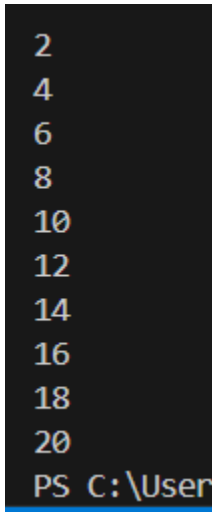
```
public class D3_7 {  
  
    public static void main(String[] args) {  
        for (int i = 2; i <= 20; i += 2) {  
            System.out.println(i);  
        }  
    }  
}
```

5. Test Cases

Present a table of test cases you used to validate your program. Include a mix of regular, boundary, and edge cases.

Test Case No.	Input	Expected Output	Actual Output	Status (Pass/Fail)
1	2,20	2,4,6,8,10,12,14,16,18,20	2,4,6,8,10,12,14,16,18,20	pass

6. Screenshots of Output



```
2
4
6
8
10
12
14
16
18
20
PS C:\User
```

7. Observation / Reflection

Loops are useful for printing sequences: The for loop in this program allows us to print a sequence of even numbers in a concise and efficient way.

Incrementing by 2 skips odd numbers: By incrementing i by 2 each time, we can skip over the odd numbers and print only the even numbers.

Simple programs can be elegant: This program demonstrates how a simple loop can be used to solve a specific problem in an elegant and efficient way.

Problem Solving Activity 3.8

1. Program Statement

Calculator Factorial for user input n . Handle edge case when $n == 0$.

2. Algorithm

1. Get the user input n .
2. Check if n is negative. If so, display an error message.
3. If n is non-negative, calculate the factorial using a loop:

- Initialize factorial to 1.
 - Iterate from 2 to n (inclusive), multiplying factorial by each number.
4. Return the calculated factorial.
-

3. Pseudocode

BEGIN

INPUT n

IF $n < 0$

PRINT "Error: Factorial is not defined for negative numbers."

ELSE

IF $n == 0$ OR $n == 1$

RETURN 1

ELSE

factorial = 1

FOR i = 2 TO n

factorial = factorial * i

END FOR

RETURN factorial

END IF

END IF

END

4. Program Code

```
import java.util.Scanner;

public class D3_8 {

    public static void main(String[] args) {
```

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

System.out.print("Enter a non-negative integer: ");

int n = scanner.nextInt();

scanner.close();

if (n < 0) {

    System.out.println("Error: Factorial is not defined for
negative numbers.");

} else {

    long factorial = calculateFactorial(n);

    System.out.println(n + "! = " + factorial);

}

}

public static long calculateFactorial(int n) {

    if (n == 0 || n == 1) {

        return 1;

    } else {

        long factorial = 1;

        for (int i = 2; i <= n; i++) {

            factorial *= i;

        }

        return factorial;

    }

}

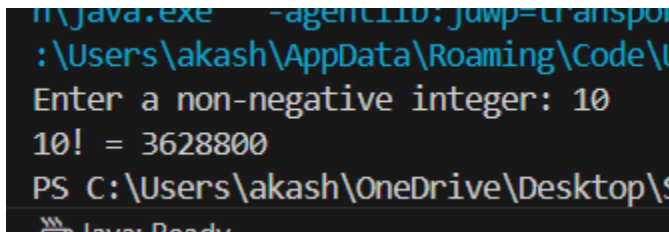
}
```

5. Test case

Present a table of test cases you used to validate your program. Include a mix of regular, boundary, and edge cases.

Test Case No.	Input	Expected Output	Actual Output	Status (Pass/Fail)
1	3	6	6	pass
2	0	1	1	pass
3	10	3628800	3628800	pass

6. Screenshots of Output



7. Observation / Reflection

Handling edge cases is important: We need to handle the edge case when $n == 0$ to ensure the program returns the correct result.

Loops are useful for calculating factorials: The loop in this program allows us to calculate the factorial of a number in a straightforward and efficient way.

Input validation is crucial: We need to validate the user input to ensure it's non-negative, as the factorial is not defined for negative numbers.

Problem Solving Activity 3.9

1. Program Statment

Ask for a string input. Count how many times 'a' or 'A' appears.

2. Algorithm

1. Get the string input from the user.
2. Convert the string to lowercase to simplify the counting process.
3. Iterate over each character in the string:
 - Check if the character is 'a'.

- If so, increment the count.

4. Return the count.

3. Pseudocode

BEGIN

INPUT str

count = 0

FOR EACH char IN str (converted to lowercase)

IF char == 'a'

count = count + 1

END IF

END FOR

RETURN count

END

4. Program Code

```
import java.util.Scanner;

public class D3_9 {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter a string: ");

        String input = scanner.nextLine();

        scanner.close();
    }
}
```

```

        int count = countA(input);

        System.out.println("The letter 'a' or 'A' appears " + count + "
times.");
    }

    public static int countA(String str) {
        int count = 0;
        for (char c : str.toLowerCase().toCharArray()) {
            if (c == 'a') {
                count++;
            }
        }
        return count;
    }
}

```

5. Test Cases

Present a table of test cases you used to validate your program. Include a mix of regular, boundary, and edge cases.

Test Case No.	Input	Expected Output	Actual Output	Status (Pass/Fail)
1	akash	The letter 'a' or 'A' appears 2 times.	The letter 'a' or 'A' appears 2 times.	
2	AkashaaA	The letter 'a' or 'A' appears 5 times.	The letter 'a' or 'A' appears 5 times.	

3	nikil	The letter 'a' or 'A' appears 0 times.	The letter 'a' or 'A' appears 0 times.	
---	-------	--	--	--

6. Screenshots of Output

```

: \Users\akash\AppData\Roaming\Code\User\work
Enter a string: AkashaaA
The letter 'a' or 'A' appears 5 times.
PS C:\Users\akash\OneDrive\Desktop\StemupBr
PS C:\Users\akash\OneDrive\Desktop\StemupBr

```

7. Observation / Reflection

Converting to lowercase simplifies the counting process: By converting the string to lowercase, we can count both 'a' and 'A' without having to check for both cases separately.

Loops are useful for iterating over strings: The loop in this program allows us to iterate over each character in the string and count the occurrences of 'a'.

Simple programs can be efficient: This program demonstrates how a simple loop can be used to solve a specific problem efficiently.

Problem Solving Activity 1.10

1. Program Statement

Simple Star Pattern Print:***** Using one for loop.

2. Algorithm

1. Determine the number of stars to print (n).
2. Use a for loop to repeat the following step n times:
 - Print a star (*).

3. Pseudocode

BEGIN

SET n = 5 (number of stars)

FOR i = 0 TO n-1

PRINT "*"

END FOR

END

4. Program Code

```
public class D3_10 {  
  
    public static void main(String[] args) {  
  
        int n = 5;  
  
        for (int i = 0; i < n; i++) {  
  
            System.out.print("*");  
  
        }  
  
    }  
  
}
```

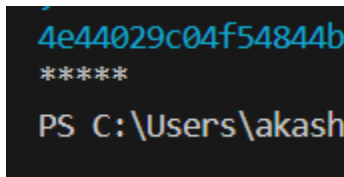
5. Test Cases

Present a table of test cases you used to validate your program. Include a mix of regular, boundary, and edge cases.

Test Case No.	Input	Expected Output	Actual Output	Status (Pass/Fail)
1	n=5	*****	*****	pass
2				

3				
---	--	--	--	--

6. Screenshots of Output



7. Observation / Reflection

Loops are useful for repetitive tasks: The for loop in this program allows us to print a sequence of stars in a concise and efficient way.

Simple programs can be elegant: This program demonstrates how a simple loop can be used to solve a specific problem in an elegant and efficient way.

Output can be controlled precisely: By using a loop, we can control the output precisely and print the desired number of stars.

Problem Solving Activity 3.11

1. Program Statement

Check if a number is prime using a loop and break.

2. Algorithm

1. Get the number to check for primality.
2. Check if the number is less than or equal to 1. If so, it's not prime.
3. Use a loop to check for divisibility from 2 to the square root of the number:

- If the number is divisible by any of these values, it's not prime.
 - If the loop completes without finding a divisor, the number is prime.
-

3. Pseudocode

BEGIN

INPUT num

IF num \leq 1

 RETURN false

END IF

FOR i = 2 TO sqrt(num)

 IF num MOD i == 0

 RETURN false

 END IF

END FOR

RETURN true

END

4. Program Code

```
import java.util.Scanner;

public class D3_11 {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter a number: ");

        int num = scanner.nextInt();

        scanner.close();
    }
}
```

```
        if (isPrime(num)) {  
            System.out.println(num + " is a prime number.");  
        } else {  
            System.out.println(num + " is not a prime number.");  
        }  
    }  
}  
  
public static boolean isPrime(int num) {  
    if (num <= 1) {  
        return false;  
    }  
    for (int i = 2; i <= Math.sqrt(num); i++) {  
        if (num % i == 0) {  
            return false;  
        }  
    }  
    return true;  
}  
}
```

5. Test Cases

Present a table of test cases you used to validate your program. Include a mix of regular, boundary, and edge cases.

Test Case No.	Input	Expected Output	Actual Output	Status (Pass/Fail)
---------------	-------	-----------------	---------------	--------------------

1	4	4 is not a prime number.	4 is not a prime number.	pass
2	7	7 is a prime number.	7 is a prime number.	pass
3	10	10 is not a prime number.	10 is not a prime number.	pass

6. Screenshots of Output

```

C:\Users\akash\AppData\Roaming
Enter a number: 10
10 is not a prime number.
PS C:\Users\akash\OneDrive\De
PS C:\Users\akash\OneDrive\De

```

7. Observation / Reflection

Loops are useful for checking primality: The loop in this program allows us to check for divisibility and determine if a number is prime.

Breaking out of a loop can improve efficiency: By returning false as soon as we find a divisor, we can avoid unnecessary iterations and improve efficiency.

Mathematical optimizations can improve performance: Checking up to the square root of the number reduces the number of iterations and improves performance.

Problem Solving Activity 3.12

1. Program Statement

Input 5 numbers. Use continue to skip negative ones and sum the rest.

2. Algorithm

1. Initialize a sum variable to 0.
2. Use a loop to input 5 numbers:

- Check if the number is negative. If so, print a message and skip to the next iteration using continue.

- If the number is non-negative, add it to the sum.

3. After the loop, print the sum of the positive numbers.

3. Pseudocode

BEGIN

sum = 0

FOR i = 1 TO 5

INPUT num

IF num < 0

PRINT "Skipping negative number: " + num

CONTINUE

END IF

sum = sum + num

END FOR

PRINT "Sum of positive numbers: " + sum

END

4. Program Code

```
import java.util.Scanner;

public class D3_12 {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        int sum = 0;

        for (int i = 0; i < 5; i++) {
```

```

System.out.print("Enter number " + (i + 1) + ": ");

int num = scanner.nextInt();

if (num < 0) {
    System.out.println("Skipping negative number: " + num);
    continue;
}

sum += num;
}

scanner.close();

System.out.println("Sum of positive numbers: " + sum);
}
}

```

5. Test Cases

Present a table of test cases you used to validate your program. Include a mix of regular, boundary, and edge cases.

Test Case No.	Input	Expected Output	Actual Output	Status (Pass/Fail)
1	3,2,-3 ,4,6	15	15	pass
2	-4,-5, -6,4,- 2	4	4	pass

3	3,4,8, -9,4	19	19	pass
---	----------------	----	----	------

6. Screenshots of Output

```
4e44029c04f54844b6b411e\redhat.java
Enter number 1: 2
Enter number 2: 3
Enter number 3: -3
Skipping negative number: -3
Enter number 4: 4
Enter number 5: 6
Sum of positive numbers: 15
PS C:\Users\akash\OneDrive\Desktop
```

7. Observation / Reflection

Continue can be used to skip iterations: The continue statement allows us to skip negative numbers and focus on summing the positive ones.

Loops can be used to process multiple inputs: The loop in this program allows us to process 5 numbers and calculate the sum of the positive ones.

Conditional statements can control program flow: The if statement in this program controls whether we skip a number or add it to the sum.

Problem Solving Activity 3.13

1. Program Statement

Input rows and cols, print a rectangle of *.

2. Algorithm

1. Get the number of rows and columns from the user.
2. Use a nested loop structure:

- The outer loop iterates over the rows.
 - The inner loop iterates over the columns and prints a * for each column.
3. After each row, print a newline character to move to the next row.
-

3. Pseudocode

BEGIN

INPUT rows

INPUT cols

FOR i = 0 TO rows-1

FOR j = 0 TO cols-1

PRINT "*" "

END FOR

PRINTLN

END FOR

END

4. Program Code

```
import java.util.Scanner;

public class D3_13 {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter number of rows: ");

        int rows = scanner.nextInt();
```

```

System.out.print("Enter number of columns: ");

int cols = scanner.nextInt();

scanner.close();


for (int i = 0; i < rows; i++) {

    for (int j = 0; j < cols; j++) {

        System.out.print("* ");

    }

    System.out.println();

}

}
}

```

5. Test Cases

Present a table of test cases you used to validate your program. Include a mix of regular, boundary, and edge cases.

Test Case No.	Input	Expected Output	Actual Output	Status (Pass/Fail)
1	4,5	***** ***** *****	***** ***** *****	pass

2				
3				

6. Screenshots of Output

```

Enter number of rows: 4
Enter number of columns: 5
* * * * *
* * * * *
* * * * *
* * * * *
PS C:\Users\akash\OneDrive\D

```

7. Observation / Reflection

Nested loops can be used to print 2D shapes: The nested loop structure in this program allows us to print a rectangle with the specified number of rows and columns.

Loops can be used to repeat tasks: The loops in this program repeat the task of printing a * for each column and row.

Program output can be formatted precisely: By using nested loops and printing newline characters, we can format the output precisely to create a rectangle shape.

Problem Solving Activity 3.14

1. Program Statement

Input height. Print right-angled triangle with *.

2. Algorithm

1. Get the height of the triangle from the user.

2. Use a nested loop structure:

- The outer loop iterates over the rows of the triangle.
- The inner loop iterates from 0 to the current row number (i) and prints a * for each iteration.

3. After each row, print a newline character to move to the next row.

3. Pseudocode

BEGIN

INPUT height

FOR i = 0 TO height-1

FOR j = 0 TO i

PRINT "*" "

END FOR

PRINTLN

END FOR

END

4. Program Code

```
import java.util.Scanner;

public class D3_14 {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter height: ");
```



```
int height = scanner.nextInt();

scanner.close();

for (int i = 0; i < height; i++) {
    for (int j = 0; j <= i; j++) {
        System.out.print("* ");
    }
    System.out.println();
}
}
```

5. Test Cases

Present a table of test cases you used to validate your program. Include a mix of regular, boundary, and edge cases.

Test Case No.	Input	Expected Output	Actual Output	Status (Pass/Fail)
1	4	* ** *** ****	* ** *** ****	pass
2				
3				

6. Screenshots of Output

```
Enter height: 5
*
* *
* * *
* * * *
* * * * *

PS C:\Users\akash\OneDrive\D
```

7. Observation / Reflection

Nested loops can be used to print triangles: The nested loop structure in this program allows us to print a right-angled triangle with the specified height.

Loop variables can be used to control output: The loop variables *i* and *j* are used to control the number of * printed in each row.

Program output can be formatted precisely: By using nested loops and printing newline characters, we can format the output precisely to create a triangle shape.

Problem Solving Activity 3.15

1. Program Statement

Input height. Print centered pyramid:

2. Algorithm

1. Get the height of the pyramid from the user.
 2. Use a nested loop structure:
 - The outer loop iterates over the rows of the pyramid.
 - The first inner loop prints spaces to center the pyramid.
 - The second inner loop prints * for each row.
 3. After each row, print a newline character to move to the next row.
-

3. Pseudocode

BEGIN

INPUT height

FOR i = 0 TO height-1

 // Print spaces

 FOR j = 0 TO height-i-2

 PRINT " "

 END FOR

 // Print stars

 FOR k = 0 TO i

 PRINT "*" "

 END FOR

PRINTLN

END FOR

END

4. Program Code

```
import java.util.Scanner;

public class D3_15 {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter height: ");

        int height = scanner.nextInt();

        scanner.close();

        for (int i = 0; i < height; i++) {

            for (int j = 0; j < height - i - 1; j++) {

                System.out.print(" ");

            }

            for (int k = 0; k <= i; k++) {

                System.out.print("* ");

            }

            System.out.println();

        }

    }

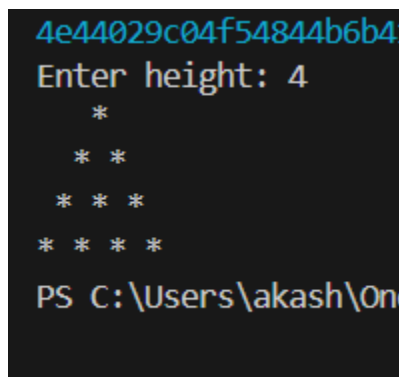
}
```

5. Test Cases

Present a table of test cases you used to validate your program. Include a mix of regular, boundary, and edge cases.

Test Case No.	Input	Expected Output	Actual Output	Status (Pass/Fail)
1	4	<pre> * * * * * * * * * *</pre>	<pre> * * * * * * * * * *</pre>	pass
2				
3				

6. Screenshots of Output



```

4e44029c04f54844b6b4...
Enter height: 4
  *
 * *
* * *
* * * *
PS C:\Users\akash\One...
```

7. Observation / Reflection

Nested loops can be used to print complex shapes: The nested loop structure in this program allows us to print a centered pyramid with the specified height.

Loop variables can be used to control output: The loop variables i, j, and k are used to control the number of spaces and * printed in each row.

Program output can be formatted precisely: By using nested loops and printing newline characters, we can format the output precisely to create a pyramid shape.

