### **Index of Activities**

1. **Swap Two Variables**
2. **Print Even Numbers with Range and Count of Even Numbers, Sum of the Even Numbers**
3. **Calculating Powers**
4. **Using Variables to Store Computation Results**
5. **Variable Reuse**
6. **Variable Shadowing**
7. **Fibonacci Series**
8. **Print Prime Numbers**
9. **Calculate Factorial Using an Iterative Approach**
10. **Find the Second Largest Number in a List**
11. **Odd or Even Checker**
12. **Grade Classification**
13. **Age Group**
14. **Leap Year Checker**
15. **Largest of Three Numbers**
16. **Positive, Negative, or Zero**
17. **Number Range Check**
18. **Check Vowel or Consonant**
19. **Simple Calculator**
20. **Password Strength Checker**
21. **Sum of First N Natural Numbers**
22. **Reverse a Number**
23. **Factorial Calculation**
24. **Fibonacci Series up to N Terms**
25. **Sum of Digits of a Number**
26. **Counting Digits in a Number**
27. **Sum of Even Numbers up to N**
28. **Printing Multiplication Table of a Number**
29. **Count Vowels in a String**
30. **Count Occurrences of a Character in a String**
31. **Right-Angle Triangle Pattern**
32. **Pyramid Pattern**
33. **Diamond Pattern**
34. **Hollow Pyramid Pattern**
35. **Floyd's Triangle**

**Python Programming Fundamentals**

#### **Activity-1: *Swap Two Variables***

**Problem Statement:** Write a Python program to swap two variables with using a temporary variable.

**Instructions:**

**Variable Initialization:** Start by initializing two variables a and b with their respective values (a = 5, b = 10 in the example).

**Display Initial Values:** Print the initial values of a and b using formatted strings to show Before swapping message.

**Swap Operation:** Use a temporary variable (temp) to store the value of a temporarily. Then, assign the value of b to a. Finally, assign the value of temp (which holds the original value of a) to b. This effectively swaps the values of a and b.

**Display Swapped Values:** Print the updated values of a and b using formatted strings to show After swapping message.

**Expected Output:**

**Ex: a=5, b=10**

Before swapping: a = 5, b = 10

After swapping: a = 10, b = 5

#### **Activity-2: *Print even Number with range and count of even numbers, sum of the even number***

**Instructions:**

* **Range of Numbers:** The program focuses on even numbers from 2 to 10.
* **Printing Even Numbers:** Using a loop, iterate through each number in the range (2 to 10). Check if the number is even (i.e., divisible by 2 without a remainder). If it is even, print the number.
* **Count of Even Numbers:** Maintain a counter that increments each time an even number is encountered and printed.
* **Sum of Even Numbers:** Initialize a variable to keep track of the sum of all even numbers. Add each even number to this variable as it is identified and printed.
* **Output Summary:** After printing all even numbers:
* Print the count of even numbers.
* Print the sum of all even numbers.

**Expected Output:**

2

4

6

8

10

Count of even numbers: 5

Sum of even numbers: 30

**Activity 3: Calculating Powers**

**Problem Statement:** Calculate and print the third power of each integer from 1 to 5.

**Instructions:**

* 1. **Initialization**: Begin by iterating through integers from 1 to 5. This can be achieved using a for loop that iterates over a range from 1 to 6 (exclusive of 6).
  2. **Cube Calculation**: For each integer in the range, calculate its cube. In Python, the cube of a number n can be calculated using the exponentiation operator \*\*. Specifically, n \*\* 3 computes the cube of n.
  3. **Output**: Print the result for each integer in the specified format. Use formatted strings (f-strings in Python) to construct the output message dynamically.

**Expected Output:**

The cube of 1 is 1

The cube of 2 is 8

The cube of 3 is 27

The cube of 4 is 64

The cube of 5 is 125

#### ***Activity 4: Using Variables to Store Computation Results***

**Problem Statement:** Use variables to store the results of arithmetic operations and print them.

**Instructions:**

**1.Variables Initialization:**

Four variables (a, b, sum\_result, difference\_result, product\_result, quotient\_result) are initialized with predefined values.

a = 15 and b = 12 are assigned initial values for performing arithmetic operations.

**2.Arithmetic Operations:**

**Sum:** Adds the values of a and b to compute the sum (sum\_result).

**Difference:** Subtracts the value of b from a to compute the difference (difference\_result).

**Product:** Multiplies the values of a and b to compute the product (product\_result).

**Quotient:** Divides the value of a by b to compute the quotient (quotient\_result).

**Expected Output:**

Sum: 27

Difference: 3

Product: 180

Quotient: 1.25

#### ***Activity 5: Variable Reuse***

**Problem Statement:** Show the effects of reusing a variable in different parts of a program.

**Instructions:**

* **Initial x: 100**
* At the beginning of the program, the variable x is initialized with the integer value 100. This is its starting point.
* **Incremented x: 120**
* Next, x is incremented by 20. So, after this operation, x becomes 120. This shows how modifying x affects its numeric value.
* **Type changed x: Reset to string**
* Finally, x is reassigned with the string value "Reset to string". This demonstrates that Python allows variables to change types during runtime. Initially, x was an integer (100), then it became an integer (120), and finally, it became a string ("Reset to string").

**Expected Output:**

Initial x: 100

Incremented x: 120

Type changed x: Reset to string

#### ***Activity 6: Variable Shadowing***

**Problem Statement:** Illustrate variable shadowing within a function.

**Instructions:**

* 1. **Global Variable (x = 10)**: At the beginning of the program, x is declared as a global variable with a value of 10.
  2. **Function Definition (shadow\_example())**: Inside the function shadow\_example(), there's a local variable x declared with a value of 5. This local variable x shadows the global variable x within the scope of the function.
  3. **Printing Local x**: When shadow\_example() is called, it prints the value of the local variable x, which is 5. This demonstrates that within the function's scope, the local variable x (with a value of 5) is used.
  4. **Printing Global x**: After calling shadow\_example(), the program prints the value of the global variable x, which remains 10. This shows that outside the function's scope, the global variable x is accessible and retains its original value.

**Expected Output:**

Local x: 5

Global x: 10

#### ***Activity 7: Fibonacci-series***

Generators are a simple way to create iterators in Python. They are particularly useful for generating sequences like the Fibonacci series.

**Instructions:**

This Python program defines a generator function fibonacci\_generator(n) that generates the first n Fibonacci numbers. Here's how the program works:

* **Initialization**: The function initializes two variables a and b to 0 and 1 respectively. These variables represent the first two numbers of the Fibonacci sequence.
* **Generator Function**: Using the yield keyword inside a loop (for \_ in range(n)), the function generates Fibonacci numbers. yield is used instead of return to create a generator that produces a sequence of values on demand.
* **Sequence Generation**: Inside the loop:
  + The current Fibonacci number a is yielded.
  + Then, a is updated to b, and b is updated to the sum of the previous a and b.
* **Printing Fibonacci Numbers**: The print(list(fibonacci\_generator(10))) statement demonstrates the usage of the generator by generating and printing the first 10 Fibonacci numbers as a list.

**Ex: [0,1, {0+1=1}, {0+1+1=2}, {0+1+1+2=3} etc...]**

**Expected Output:**

[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]

#### ***Activity 8: Print Prime Numbers with range 1,10***

**Instructions:**

* **Define the Range:** First, define the range of numbers from 1 to 10.
* **Prime Number Check:** Iterate through each number in the defined range and check if it is a prime number.
* A prime number is only divisible by 1 and itself.
* **Print Prime Numbers:** For each number that satisfies the prime condition, print it.

**Expected Output:**

2

3

5

7

#### ***Activity 9****:* ***Calculate factorial using an iterative approach.***

**Instructions:**

Factorial is the product of all positive integers less than or equal to a given positive integer. The iterative approach involves using a loop to multiply numbers sequentially from 1 up to the given number to compute its factorial.

**Input:**

Prompt the user to enter a number (num) for which they want to calculate the factorial.

**Factorial Calculation:**

Initialize a variable (factorial) to store the factorial value, starting with 1.

Use a for loop to iterate from 1 to num (inclusive).

Inside the loop, multiply factorial by each integer from 1 to num.

After completing the loop, factorial will contain the factorial of num.

**Output:**

Print the computed factorial value.

**Expected Output:**

**Enter a num:5**

120 ---->(5\*4\*3\*2\*1)

#### ***Activity 10: Find the second largest number in a list.***

**Instructions:**

The function second\_largest(lst) takes a list lst as input and performs the following steps:

**Remove Duplicates**: It converts the list lst into a set using set(lst), which automatically removes duplicate elements. Then, it converts the set back into a list unique\_numbers.

**Sort the List**: The function sorts the unique\_numbers list in ascending order using unique\_numbers.sort(). This ensures that the largest numbers are at the end of the list after sorting.

**Return Second Largest**: Finally, the function returns the second last element of the sorted unique\_numbers list using unique\_numbers[-2]. Since the list is sorted in ascending order, unique\_numbers[-1] gives the largest number, and unique\_numbers[-2] gives the second largest number.

**List= [1, 3, 5, 7, 7, 8, 8, 10]**

**Expected Output**: 8

### **If ... Else**

**Activity 11: Odd or Even Checker**

**Instructions:**

* **Input Prompt**: The program prompts the user to enter a number. For example, if the user enters 15, the program expects this input.
* **Reading Input**: It reads the number entered by the user and stores it in a variable.
* **Checking Odd or Even**: It then checks whether the number is odd or even using a simple condition:
* If the number modulo 2 (number % 2) equals 0, the number is even.
* If number % 2 does not equal 0, the number is odd.
* **Output**: Based on the condition check, the program prints "Even" if the number is even and "Odd" if the number is odd.

**Expected Output:**

**Enter a number: 15**

15 is odd.

**Activity 12: Grade Classification**

**Instructions:**

Write a Python program that determines the letter grade based on a numeric grade input. The program should output the corresponding letter grade according to the following criteria:

* If the grade is 90 or above, output "A".
* If the grade is between 80 and 89 (inclusive), output "B".
* If the grade is between 70 and 79 (inclusive), output "C".
* If the grade is between 60 and 69 (inclusive), output "D".
* If the grade is below 60, output "F".

**Expected Output:** B

**Activity 13: Age Group**

**Instructions:**

Write a Python program that categorizes a person based on their age into one of the following categories: Child, Teenager, Adult, or Senior. The program should output the appropriate category based on the following criteria:

* If the age is less than 13, output "Child".
* If the age is between 13 and 19 (inclusive), output "Teenager".
* If the age is between 20 and 59 (inclusive), output "Adult".
* If the age is 60 or above, output "Senior".

Implement the program using conditional statements (if, elif, else) to determine the category based on the provided age. Ensure that your program correctly categorizes the age based on the specified ranges.

**Expected Output:** Adult

**Activity 14: Leap Year Checker**

**Instructions:**

**Input:**

The program takes an input year from the user or from predefined variables in the code.

**Leap Year Criteria:**

A year is considered a leap year if:

It is divisible by 4,

Except for years divisible by 100, unless they are also divisible by 400.

**Implementation:**

The program first prompts the user for input or uses a predefined year.

It checks whether the year satisfies the leap year conditions using conditional statements (if, elif, else).

If the year meets the criteria, it prints that the year is a leap year.

If not, it prints that the year is not a leap year.

**Example:**

For example, the year 2000 is a leap year because it is divisible by 400.

The year 1900 is not a leap year because it is divisible by 100 but not by 400.

The year 2024 is a leap year because it is divisible by 4.

**Expected Output:** 2024 is a leap year.

**Activity 15: Largest of Three Numbers**

**Instructions:**

The goal of this Python program is to determine the largest number among three given numbers (a, b, c). The program uses conditional statements to compare these numbers and assigns the largest number to the variable largest. Here's a breakdown of how the program works:

**Initialization:**

Three variables a, b, and c are initialized with the values 5, 10, and 3 respectively. These represent the numbers among which we want to find the largest.

**Comparison Logic:**

The program uses nested if, elif, and else statements to compare a, b, and c.

The first if statement checks if a is greater than or equal to both b and c. If true, it assigns a to largest.

The first elif statement checks if b is greater than or equal to both a and c. If true, it assigns b to largest.

If neither of the above conditions is true (meaning c is the largest or tied with another number), the else statement assigns c to largest.

**Output:**

Finally, the program prints out the result using an f-string (f"The largest number is {largest}."). This displays the largest number among a, b, and c based on the comparisons made.

**Expected Output:** The largest number is 10.

**Activity 16: Positive, Negative, or Zero**

**Instructions:**

* **Input:** The program takes a numeric input from the user or uses a predefined value assigned to a variable.
* **Conditional Checks:**
* If the number is greater than 0 (number > 0), it prints "Positive".
* If the number is less than 0 (number < 0), it prints "Negative".
* If the number equals 0 (number == 0), it prints "Zero".
* **Output:** Depending on the value of the input number, the program outputs one of the following messages:
* "Positive" if the number is greater than 0.
* "Negative" if the number is less than 0.
* "Zero" if the number is equal to 0.

**Activity 17: Number Range Check**

**Instructions:**

The program evaluates whether a given number (number) falls within the range of 0 to 100. It uses a conditional statement (if and else) to determine and print a message based on the number's range.

**Expected Output:** Number is within the range 0-100.

**Activity 18: Check Vowel or Consonant**

**Instructions:**

This Python program checks whether a given character (char) is a vowel or a consonant. It uses a conditional statement (if) to determine if the character belongs to the set of vowels ('a', 'e', 'i', 'o', 'u'). If the character matches any of these vowels, it prints a message indicating that the character is a vowel. Otherwise, it assumes the character is a consonant and prints a message stating that it is not a vowel.

**Expected Output:** a is a vowel.

**Activity 19: Simple Calculator**

**Instructions:**

**Variables:**

* num1, num2: These variables are initialized with values 10 and 5 respectively, representing the operands of the calculator.
* operation: This variable is initialized with the value '+', indicating the operation to be performed (addition in this case).

**Operations:**

* The program checks the value of operation using conditional statements (if, elif, else) to determine which arithmetic operation to perform.
* Depending on the value of operation, one of the following calculations is executed:
  + If operation is '+', it computes the sum of num1 and num2.
  + If operation is '-', it computes the difference between num1 and num2.
  + If operation is '\*', it computes the product of num1 and num2.
  + If operation is '/', it computes the division of num1 by num2.
  + If operation does not match any of these cases, it assigns "Invalid operation" to the result variable.

**Output:**

* After performing the calculation based on the value of operation, the program prints the result using an f-string (f"Result: {result}"). If operation is '+' with num1 as 10 and num2 as 5, the expected output is Result: 15.

**Expected Output:** Result: 15

**Activity 20: Password Strength Checker**

**Instructions:**

**Expected Output:** Strong password.

This program is designed to evaluate the strength of a given password based on certain criteria. The goal is to assess how secure a password is, typically for use in enhancing cybersecurity measures.

**Instructions:**

**Input Password:** Users will provide a password as input to the program.

**Criteria Evaluation:**

**Length:** The program checks the length of the password. Typically, longer passwords are considered stronger.

**Complexity:** It evaluates whether the password includes a mix of different character types, such as uppercase letters, lowercase letters, numbers, and special characters.

**Commonality:** It may also check if the password contains common patterns or easily guessable sequences, like "12345" or "password".

**Dictionary Check:** Some implementations may check if the password appears in a dictionary of commonly used passwords, indicating vulnerability.

**Output Strength Level:** Based on these evaluations, the program provides feedback on the strength of the password. This feedback could be qualitative (e.g., "Weak", "Medium", "Strong") or quantitative (e.g., a score out of 10).

**Recommendations:** Depending on the implementation, the program may suggest improvements to the password, such as increasing length, adding special characters, or avoiding common words.

**Security Considerations:** The program should be designed with security in mind, handling user input safely to prevent vulnerabilities like injection attacks or unintended code execution.

**Usage Context:** This program is useful in various contexts where password security is critical, such as user account management systems, online banking, or any application handling sensitive information.

**Educational Purpose:** It can also serve an educational purpose by teaching users about good password practices and the importance of strong passwords in safeguarding personal and organizational data.

### **While Loops**

**Activity 21: Sum of First N Natural Numbers**

**Instructions:**

* **Initialization**: Start by initializing two variables:
* N: Represents the number of natural numbers we want to sum.
* sum: Initializes to 0 to store the cumulative sum of natural numbers.
* **Input**: The program does not take direct user input for *NN*N in this case; instead, it's predefined as 10. You can modify N to calculate the sum for any number of natural numbers you want.
* **While Loop**: Use a while loop to iterate from 1 through *NN*N:
* **Condition**: The loop continues as long as a counter variable, num, is less than or equal to N.
* **Summation**: Inside the loop, add the current value of num to sum in each iteration.
* **Increment**: Also, increment the num by 1 in each iteration to move to the next natural number.
* **Output**: Once the loop completes, print the result using formatted string output that displays the total sum of the first *NN*N natural numbers.

**Expected Output:** Sum of first 10 natural numbers is 55.

**Activity 22: Reverse a Number**

**Instructions:**

To reverse a given number in Python, you can follow these steps:

1. **Input the Number**: First, input the number you want to reverse. Let's say the number is 12345.
2. **Initialize Variables**: Create variables to store the original number (number) and the reversed number (reversed\_number). Initially, set reversed\_number to 0.
3. **Reverse the Number**:
   * Loop through each digit of the original number using a while loop.
   * Extract the last digit of the number using the modulo operator (%) and append it to the reversed\_number.
   * Remove the last digit from number using integer division (//).
   * Repeat this process until all digits of number are processed.
4. **Output the Result**: Print the reversed\_number, which now contains the original number reversed.

**Expected Output:** Reversed number is 54321.

**Activity 23: Factorial Calculation**

**Instructions:**

This Python program calculates the factorial of a number using a while loop. Factorial of a number *nn*n, denoted as *n!n!*n!, is the product of all positive integers less than or equal to *nn*n. For example, factorial of 5 (*5!5!*5!) is calculated as *5×4×3×2×15 \times 4 \times 3 \times 2 \times 1*5×4×3×2×1, which equals 120.

**Expected Output:** Factorial of 5 is 120.

**Activity 24: Fibonacci Series up to N Terms**

**Instructions:**

The Fibonacci sequence is a series of numbers where each number is the sum of the two preceding ones, usually starting with 0 and 1. In this Python program, we generate the Fibonacci series up to N terms using a while loop.

**Initialization:**

Two variables a and b are initialized to 0 and 1 respectively. These represent the first two numbers in the Fibonacci sequence.

**Input:**

The variable N specifies the number of terms in the Fibonacci series that we want to generate. It is set by the user or defined within the program.

**Generating Fibonacci Series:**

A while loop is used to generate and print the Fibonacci series up to N terms.

Inside the loop:

The current Fibonacci number a is printed.

Then, a is updated to b, and b is updated to the sum of the previous a and b.

This process continues until the number of terms printed reaches N.

**Output:**

The Fibonacci series is printed in a single line, with each number separated by a space.

**Expected Output:** 0 1 1 2 3 5 8 13 21 34

**Activity 25: Sum of Digits of a Number**

**Instructions:**

The program calculates the sum of digits for a given number using a while loop and integer arithmetic operations.

**Initialization:**

number = 1234: This initializes the variable number with the value 1234. You can replace this value with any positive integer for which you want to calculate the sum of digits.

sum\_of\_digits = 0: This initializes sum\_of\_digits to 0. This variable will store the cumulative sum of the individual digits extracted from number.

**While Loop:**

while number > 0:: This starts a while loop that continues as long as number is greater than 0.

sum\_of\_digits += number % 10: Inside the loop, number % 10 gives the last digit of number (e.g., 1234 % 10 = 4). This digit is added to sum\_of\_digits.

number //= 10: After extracting the last digit, number is updated using integer division by 10 (number //= 10), which removes the last digit from number. This process continues until all digits are processed.

**Output:**

print(f"Sum of digits is {sum\_of\_digits}."): Once the while loop completes (when number becomes 0), the program prints the total sum of digits stored in sum\_of\_digits.

**Expected Output:** Sum of digits is 10.

**Activity 26: Counting Digits in a Number**

**Instructions:**

Write a Python program that counts the number of digits in a given integer number.

**Steps to Build the Code:**

**Initialize Variables:**

Create a variable to hold the number whose digits need to be counted (number).

Initialize a counter variable (count) to keep track of the number of digits.

**Use a While Loop:**

Start a while loop that continues as long as number is greater than 0.

Inside the loop:

Increment the count variable by 1 to count each digit.

Divide the number by 10 using integer division (//) to remove the last digit from number. This effectively reduces the number's length by one digit in each iteration.

**Output the Result:**

After the while loop completes (when number becomes 0), print the final count of digits using an f-string to format the output.

**Expected Output:** Number of digits is 6.

**Activity 27: Sum of Even Numbers up to N**

**Instructions:**

**Objective:** The program aims to compute the sum of all even numbers from 1 up to a specified number *NN*N.

**Steps and Explanation:**

**Input:**

The user provides a number *NN*N (in this case, *N=10N = 10*N=10).

**Initialization:**

Initialize a variable sum\_even to store the cumulative sum of even numbers, starting from 0 (sum\_even = 0).

**Loop through Numbers:**

Use a loop to iterate through each number from 1 to *NN*N.

Check if the current number is even (number % 2 == 0). If true, add it to sum\_even.

**Output:**

After completing the loop, print the result in the format: "Sum of even numbers up to *NN*N is *sumevensum\_even*sume ven".

**Expected Output:** Sum of even numbers up to 10 is 30.

**Activity 28: Printing Multiplication Table of a Number**

**Instructions:**

This Python program prints the multiplication table for a given number. In this example, the number chosen is 5.

**Expected Output Explanation:**

The output consists of 10 lines, each representing a multiplication operation of 5 with numbers from 1 to 10. Each line follows the format 5 x n = result, where n is the number from 1 to 10, and result is the product of 5 multiplied by n.

**Detailed Steps:**

1. Initialize a variable number with the value 5. This represents the number for which we want to generate the multiplication table.
2. Use a for loop that iterates through the range from 1 to 11 (exclusive). This range ensures that the loop runs 10 times, corresponding to the multiplication from 1 to 10.
3. Inside the loop, calculate the product of number (which is 5) and the current loop variable (i). Store this product in a variable called result.
4. Print each multiplication operation in the format "5 x i = result", where i is the current loop variable and result is the calculated product.

**Expected Output:**

5 x 1 = 5  
5 x 2 = 10  
5 x 3 = 15  
...  
5 x 10 = 50

**Activity 29: Count Vowels in a String**

**Instructions:**

This Python program counts the number of vowels in a given string (string) using a while loop. It starts by initializing a counter (count) to zero and an index (i) to zero to iterate through each character of the string. The program then enters a while loop that continues as long as i is less than the length of the string (len(string)).

Inside the loop:

* It checks if the character at the current index (string[i]) exists in the string vowels which contains all vowel characters ("a", "e", "i", "o", "u").
* If the current character is found in vowels, it increments the count variable by 1.
* Regardless of whether the character is a vowel or not, it increments the index i to move to the next character in the string.

Once the loop completes:

* It prints the total count of vowels found in the string using an f-string formatted string.

**Expected Output:** Number of vowels in the string is 3.

**For Loops**

**Activity 30:** **Count Occurrences of a Character in a String**

**Instructions:**

**Objective:**

Create a Python script that counts the occurrences of vowels in a user-provided string.

**Steps to Implement:**

1. **Input Prompt:**
2. Ask the user to input a string.
3. **Define Vowels:**

Define a string containing all vowels (aeiou).

1. **Initialize Count:**

Initialize a counter variable to store the count of vowels found.

1. **Iterate through the String:**

Use a loop to iterate through each character in the input string.

**Expected Output:** 'a' occurs 3 times in the string.

**Activity 31:** **Right-Angle Triangle Pattern**

**Instructions:**

* **Function Definition (print\_triangle):**
* print\_triangle(rows) is a function that takes one parameter rows, which specifies the number of rows (and columns) in the right-angle triangle.
* **Nested Loops:**
* The function uses nested for loops to construct the triangle:
  + The outer loop (for i in range(1, rows + 1)) iterates through each row of the triangle, from 1 to rows.
  + The inner loop (for j in range(1, i + 1)) iterates through each column in the current row, where i is the current row number.
* **Printing Asterisks (\*):**
* Inside the inner loop, print("\*", end=" ") prints an asterisk followed by a space (" "). This forms each segment of the triangle in the current row.
* **Newline (print()):**
* After printing all asterisks for the current row (inside the inner loop), print() with no arguments moves to the next line, ensuring the next row starts on a new line.
* **Example Usage:**
* The example usage section demonstrates how to call print\_triangle(rows) to print a right-angle triangle with 5 rows. Adjusting the rows variable allows you to change the size of the triangle.

**Expected Output:**

**\***   
**\* \***   
**\* \* \***   
**\* \* \* \***   
**\* \* \* \* \***

**Activity 32:** **Pyramid Pattern**

**Instructions:**

* **Input:**
* The program takes an input num\_rows, which determines the number of rows in the pyramid. For example, num\_rows = 5 would generate a pyramid with 5 rows.
* **Structure of the Pyramid:**
* Each row i in the pyramid consists of:
  + (num\_rows - i - 1) spaces to center-align the asterisks.
  + 2 \* i + 1 asterisks (\*), which increases with each subsequent row.
* **Implementation:**
* The program uses nested loops to iterate through each row and column to construct the pyramid pattern.
* The outer loop (i) iterates over each row, from 0 to num\_rows - 1.
* The inner loop (j) manages the printing of spaces and asterisks within each row based on the calculated number of spaces and asterisks.
* **Output:**
* After constructing each row, the program prints the pattern line by line, resulting in a pyramid-shaped output.

**Expected Output:**

**\***   
 **\* \* \***   
 **\* \* \* \* \***   
 **\* \* \* \* \* \* \***   
**\* \* \* \* \* \* \* \* \***

**Activity 33:** **Diamond Pattern**

**Instructions:**

The diamond pattern is a geometric pattern that resembles the shape of a diamond when printed. It consists of lines that widen in the middle and then narrow symmetrically towards the bottom and top.

**Pattern Characteristics:**

The pattern is centered both horizontally and vertically.

It consists of lines that alternate between spaces and asterisks (\*).

The widest part of the diamond occurs at the middle row.

The number of asterisks on each line decreases symmetrically as you move away from the middle row.

**Steps to Generate the Diamond Pattern:**

**Input the Number of Rows (N):** Determine how many rows the diamond pattern should have. This determines the overall height of the diamond.

**Calculate the Middle Row:** The middle row is where the diamond is widest. It occurs at row N // 2 + 1 (assuming N is odd for perfect symmetry).

**Generate the Upper Half of the Diamond:**

For rows above the middle row (from 1 to N // 2):

Calculate the number of spaces and asterisks (\*) on each line to achieve the diamond shape.

Print each line, ensuring it is centered horizontally.

**Generate the Lower Half of the Diamond:**

For rows below the middle row (from N // 2 + 1 to N):

Reflect the upper half by printing lines in reverse order (from N down to N // 2 + 1).

Ensure each line is centered horizontally and follows the same spacing and asterisk pattern as the upper half.

**Display the Diamond Pattern:** Print each line of the pattern according to the calculated spacing and asterisk distribution.

**Expected Output:**

**\***   
 **\* \* \***   
 **\* \* \* \* \***   
 **\* \* \* \* \* \* \***   
**\* \* \* \* \* \* \* \* \***   
 **\* \* \* \* \* \* \***   
 **\* \* \* \* \***   
 **\* \* \***   
 **\***

**Activity 34:** **Hollow Pyramid Pattern**

**Instructions:**

* A hollow pyramid pattern consists of asterisks (\*) arranged in a specific shape that resembles an upside-down triangle. Unlike a solid pyramid, a hollow pyramid has only the outline, with the inside empty or hollow.

### **Steps to Build the Hollow Pyramid Pattern**

* **Input the Number of Rows (Height of the Pyramid):**

You need to decide how many rows (or levels) you want your hollow pyramid to have. This determines the overall height and width of the pyramid.

* **Outer and Inner Loop Structure:**
  + Use nested loops to iterate through each row and each column within that row.
  + The outer loop controls the rows of the pyramid.
  + The inner loop controls the columns within each row.
* **Pattern Logic:**
  + Determine the number of spaces and asterisks (\*) to print in each row based on its position.
  + The top row (1st row) has the maximum number of asterisks, while the bottom row (last row) has the minimum.
  + For each row i from 1 to the total number of rows:
  + Calculate the number of spaces to print before the first asterisk to center-align the pyramid.
  + Determine which positions should print an asterisk and which should print spaces to create the hollow effect:
  + The first and last positions in each row always print an asterisk to form the pyramid's outline.
  + The positions between the first and last asterisks print spaces to hollow out the inside of the pyramid.
* **Printing the Pyramid:**
  + Construct each row by appending spaces and asterisks based on the pattern logic.
  + Print each constructed row sequentially to form the complete hollow pyramid pattern.

**Expected Output:**

**\***   
 **\* \***   
 **\* \***   
 **\* \***   
**\* \* \* \* \* \* \* \* \***

**Activity 35:** **Floyd's Triangle**

Floyd's Triangle is a right-angled triangular array of natural numbers where:

* The first row contains 1.
* The second row contains 2, 3.
* The third row contains 4, 5, 6.
* And so on.

Each row *nn*n in Floyd's Triangle contains *nn*n numbers, starting from 1 and increasing sequentially.

**Expected Output:**

Enter the number of rows: 5

1   
2 3   
4 5 6   
7 8 9 10   
11 12 13 14 15