|  |
| --- |
| Atmiya University, Rajkot, Rajkot ... |
| Cryptography and network security  Assignment 1 |
| |  |  |  | | --- | --- | --- | | Mihir Soni | 1/17/25 | 23BTITCC604 | |

**ATMIYA UNIVERSITY**

**FACULTY OF FOET**

**INFORMATION TECHNOLOGY DEPARTMENT**



A Practical Document On

**“Assignment 1”**

Under subject of

**Cryptography and network security**

Semester-6

Submitted by:

**Patadiya Mihir (220004035)**

Name of Mentor

(**Mr. Piyush Kashiyani**)

(Head of the Department)

**Mr. Darshan Jani**

A logo with text on it

Description automatically generated

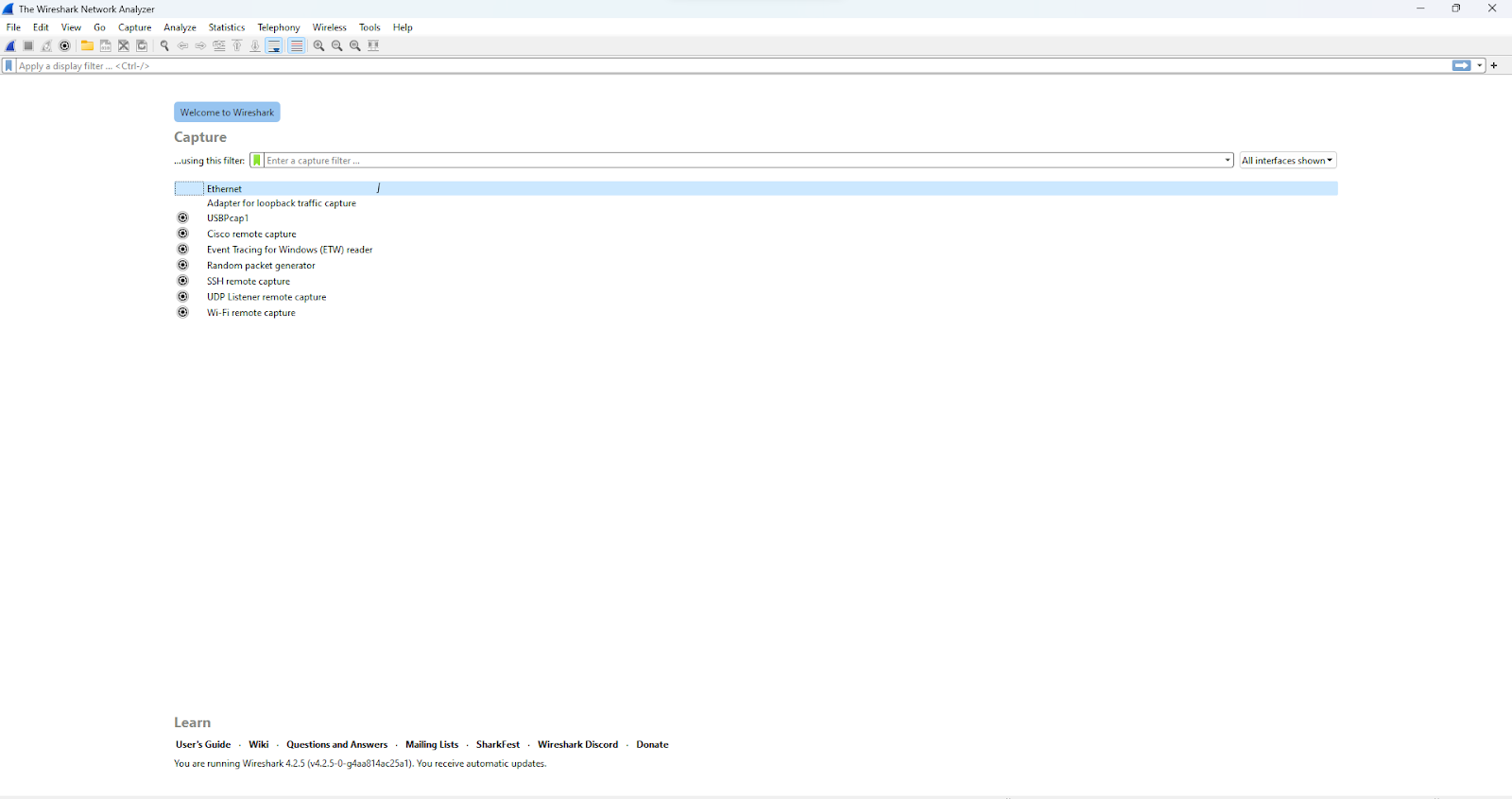
|  |  |  |
| --- | --- | --- |
| **Sr No** | **List of Practical** | Pg No. |
| **1** | Study and use the Wireshark for the various network protocols | 5 |
| **2** | WAP to find the factorial of the given number | 11 |
| **3** | WAP to find the given number is Armstrong or not.? | 12 |
| **4** | WAP to find if a given number is palindrome or not.? | 13 |
| **5** | WAP to find the given number is prime or not. | 14 |
| **6** | WAP to find the prime number from the given range by user | 15 |
| **7** | WAP to find the factors of given number (ex - 6 -> 1,2,3,6) | 16 |
| **8** | WAP to print given number into word (ex - 25 -> twenty-five) | 17 |
| **9** | Write a C Program to Print the Multiplication Table of given number | 19 |
| **10** | Write a program to reverse any number using loop | 20 |
| **11** | Write a Program to [Count the number of the digits of an Integer.](https://www.programmingwithbasics.com/2016/03/c-program-to-count-number-of-digits-of.html) | 21 |
| **12** | Write a Program to find the sum of two Matrix | 22 |
| **13** | Write a Program to find the Multiplication of two Matrix | 24 |
| **14** | Write a Program to remove all the Vowels from the String | 26 |
| **15** | Write a Program to remove the Duplicate element from the string | 27 |
| **16** | Write a Program to check given string in Camel Case or Not.? | 28 |
| **17** | WAP to find indices of array – Input = [2,2,3] sum = 5 then O/P = [1,2] indices | 29 |
| **18** | WAP to convert Roman to Integer Number | 30 |
| **19** | WAP to find Longest common prefix Input = [“Flower”,” Flow”, “Flight”]  O/P = “Fl” | 32 |
| **20** | WAP to find permutation of given number I/P – [1,2,3] to all possibilities | 33 |
| **21** | WAP to count and say I/P – 332251 -> two 3’s, two 2’s, one 5 one 1’s | 35 |
| **22** | WAP to rotate 3\*3 Matrix from row to column | 37 |
| **23** | WAP to draw Square Hollow pattern | 38 |
| **24** | WAP to print the Reverse pattern.         1      2 2   3 3 3 | 39 |
| **25** | WAP for given Pattern.  1 2 3 4 5  1 2 3 4  1 2 3  1 2    1 | 40 |
| **26** | WAP for given Triangle Pattern.    1   2 3   4 5 6  7 8 9 10 | 41 |
| **27** | WAP to print the given pattern.  1  1 2    1 2 3 | 42 |
| **28** | WAP to print the given pattern.  **\***  **\*\*\***  **\*\*\*\*\***  **\*\*\*\*\*\*\*** | 43 |
| **29** | WAP to print the given pattern.  **\***  **\* \***  **\*   \***  **\*     \***  **\*       \***  **\*         \***  **\*       \***  **\*     \***  **\*   \***  **\* \***  **\*** | 44 |
| **30** | WAP to print Right Pascal’s Triangle pattern | 46 |
| **31** | WAP to print Spiral Matrix output, starting from left to right row wise. | 47 |
| **32** | WAP to remove duplicate elements from the given elements | 49 |
| **33** | WAP to check whether a given string is isomorphic or not.? | 50 |
| **34** | WAP to Replace a character at a specific index in a String | 52 |
| **35** | WAP to Swap Two Strings without Third String Variable | 53 |
| **36** | WAP to Count Number of Words in String | 54 |
| **37** | WAP to find specific word from given string | 55 |
| **38** | WAP to find the GCD of the inputted 2 numbers | 56 |
| **39** | WAP to sort elements in given Array | 57 |
| **40** | WAP to find the Reverse of the Array | 58 |

**ANSWERS**

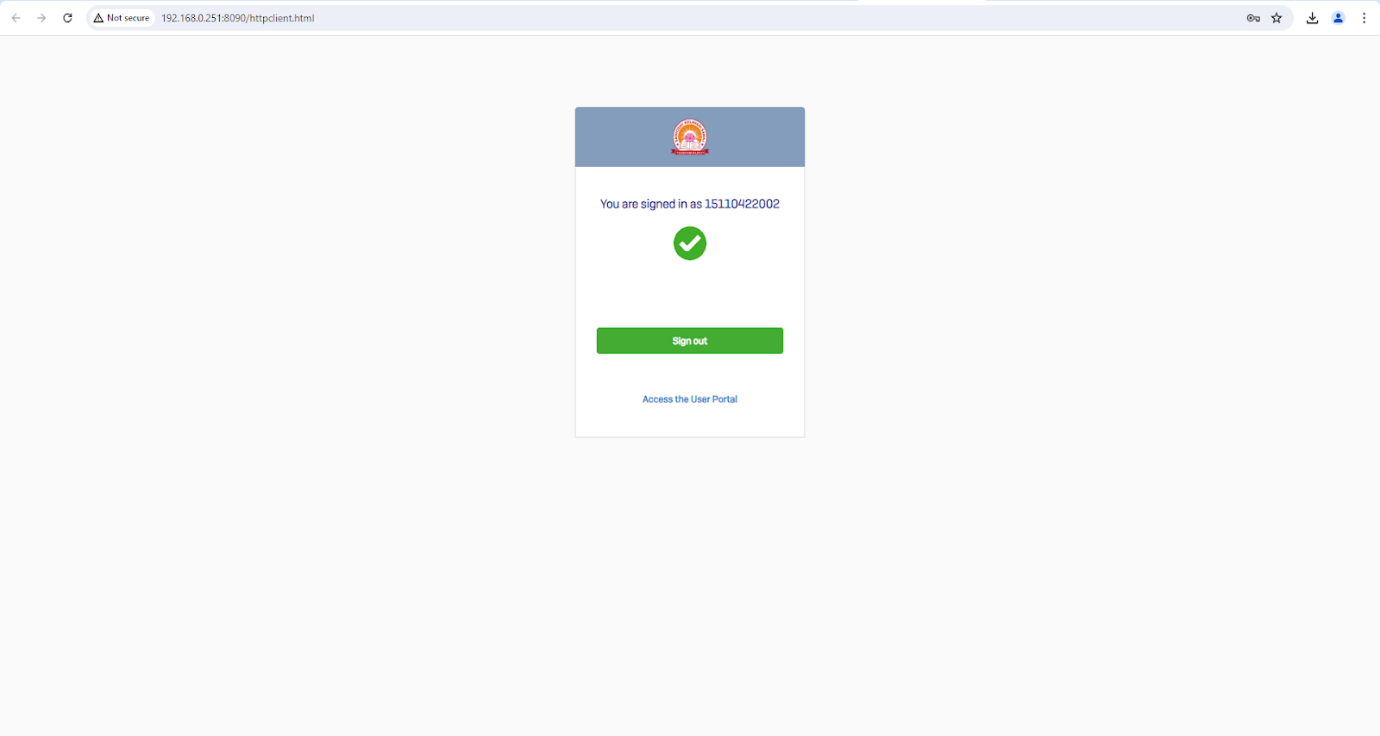
1. **Study and use the Wireshark for the various network protocols**

🡺 Steps: -

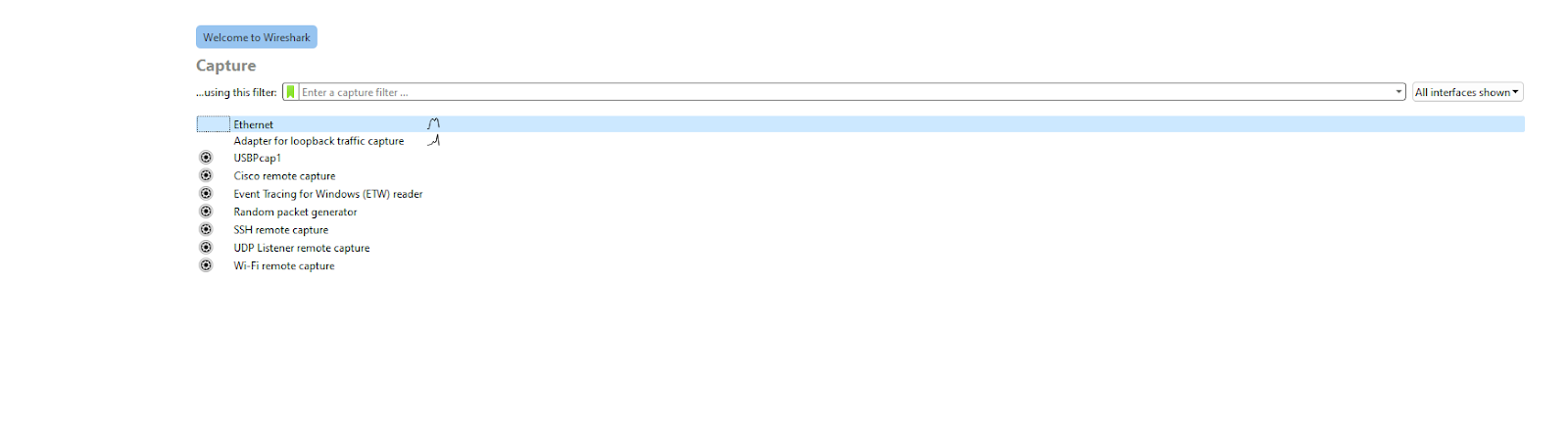
**Step 1: - Open the Wireshark App.**



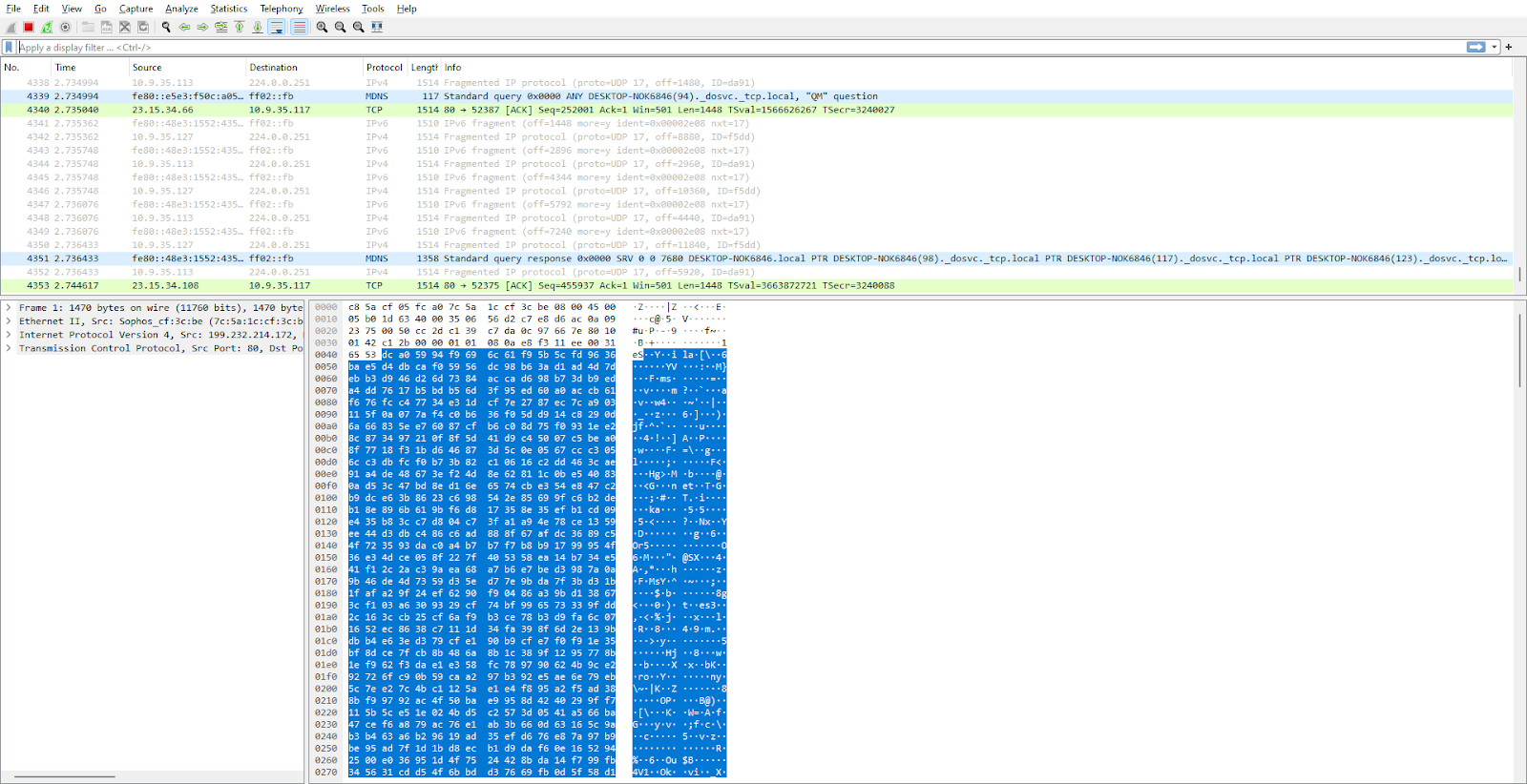
**Step 2: - Search for any unsecure site and paste the id to the Wireshark.**



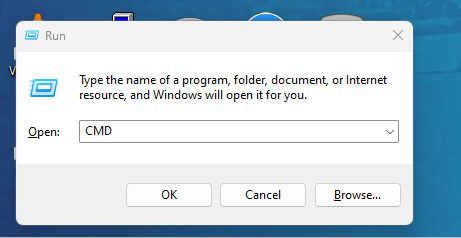
**Step 3: - Now click on the Ethernet option available here and see the processes.**



**Step 4: - Now this interface will open.**



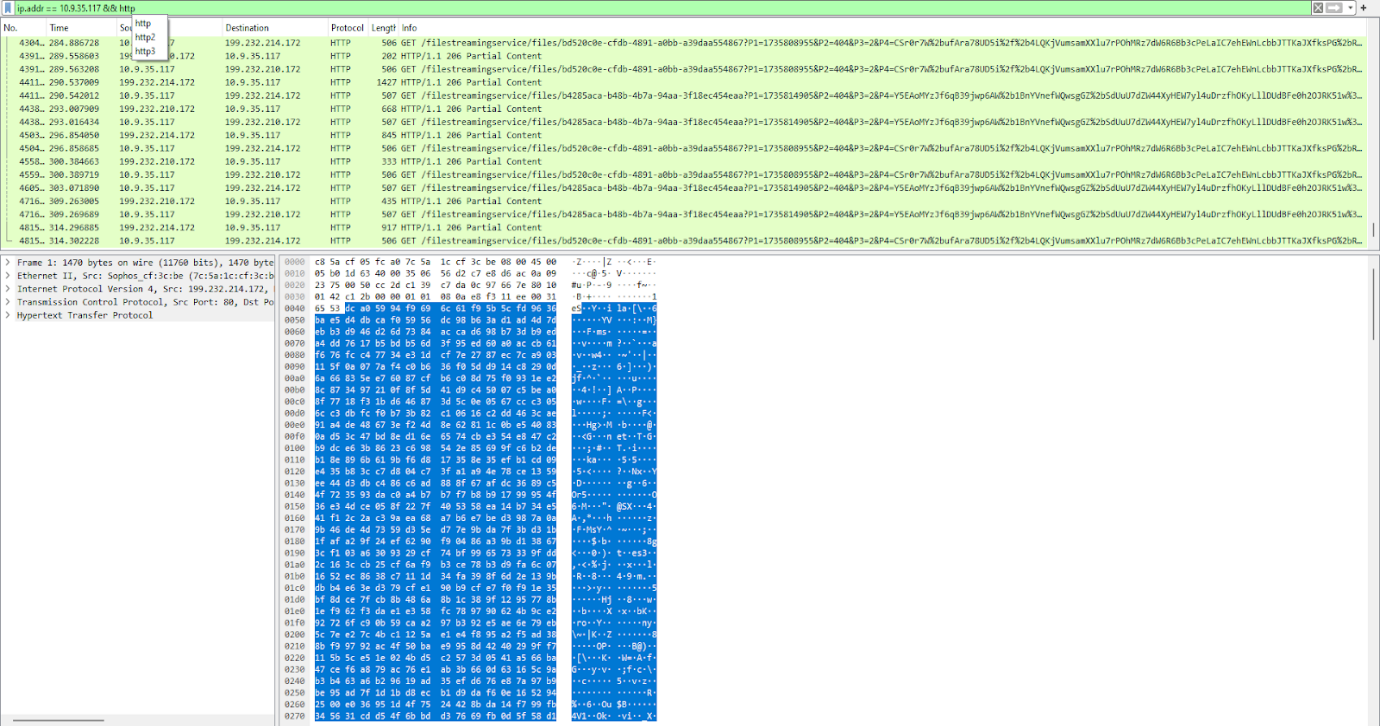
**Step 5: - Now run the Windows + R command and type in there CMD.**



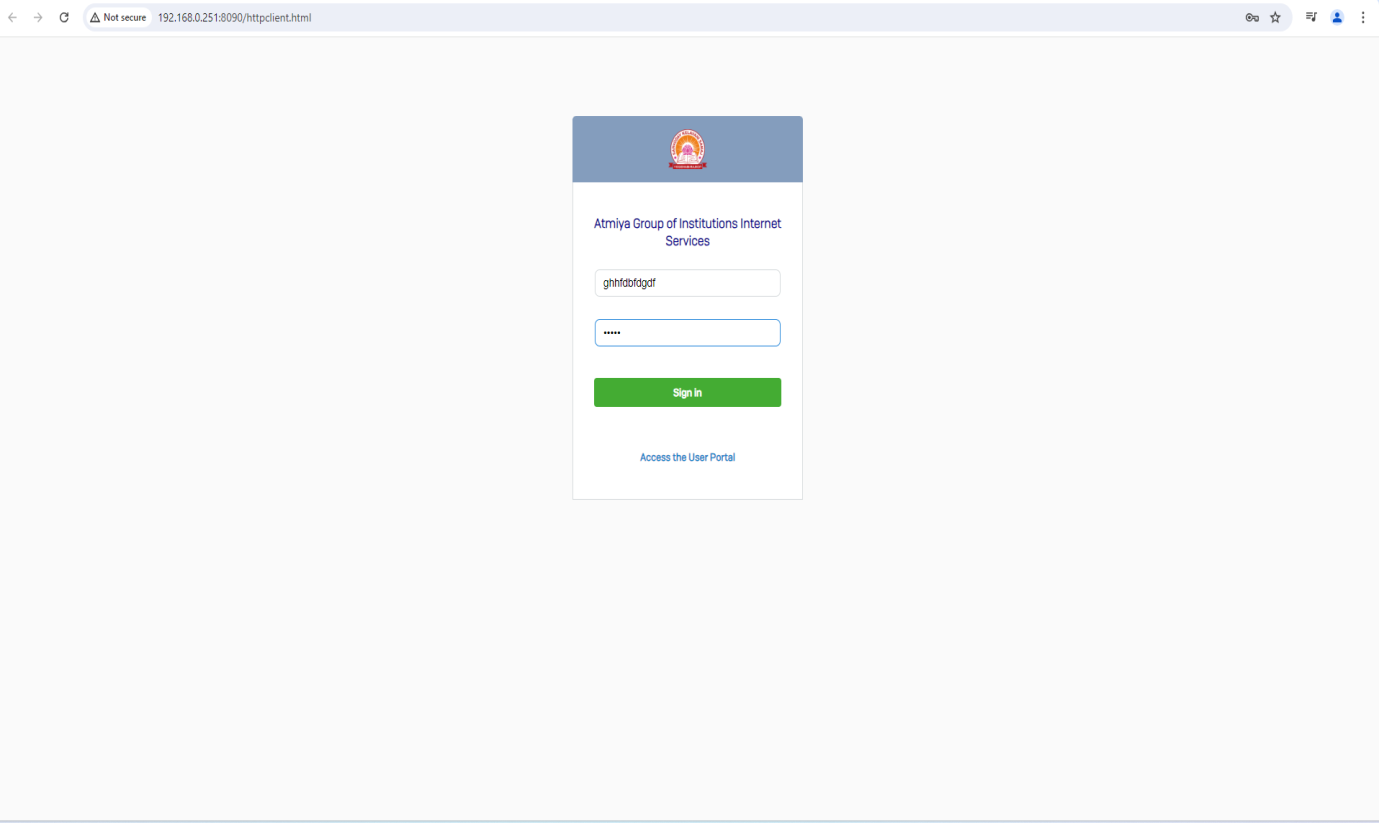
**Step 6: - Afterwards in CMD type ipconfig and note down your IP Address.**



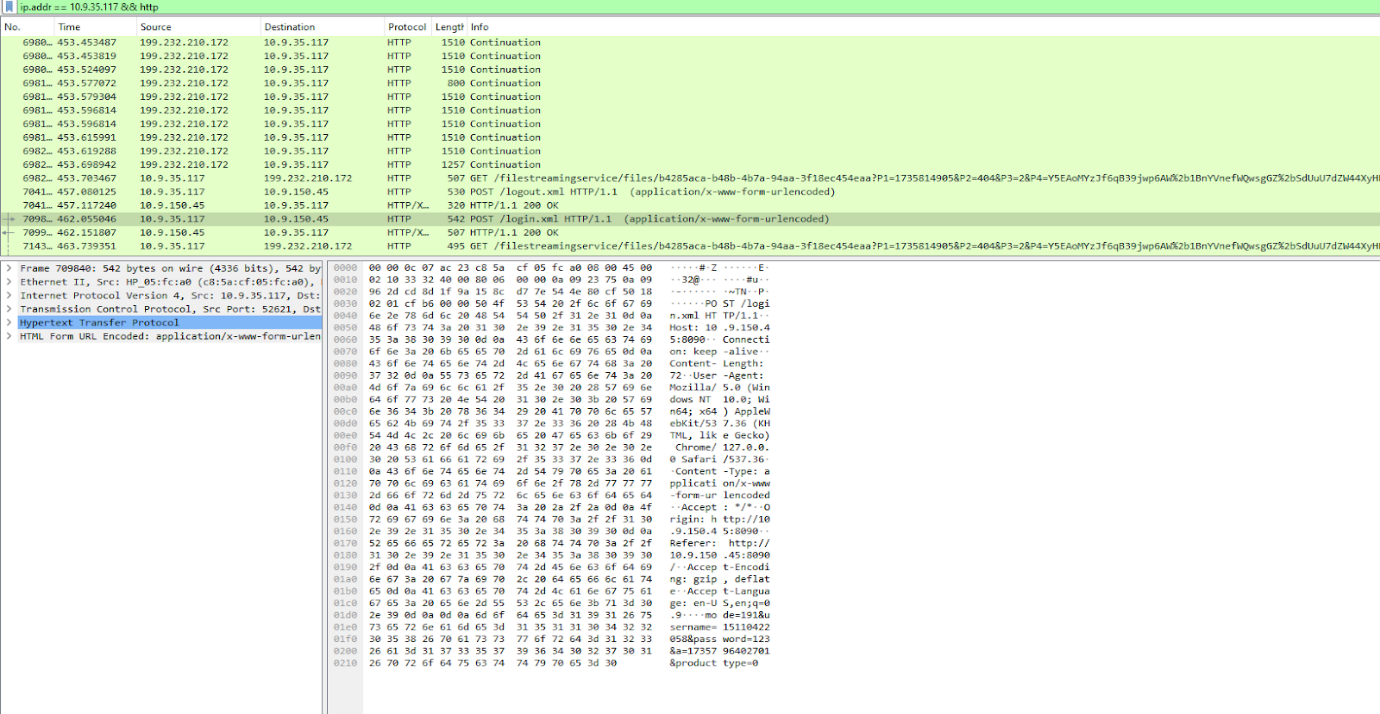
**Step 7: - Now in the filter tab write down your IP and mention the HTTP protocol and you will see the required result.**



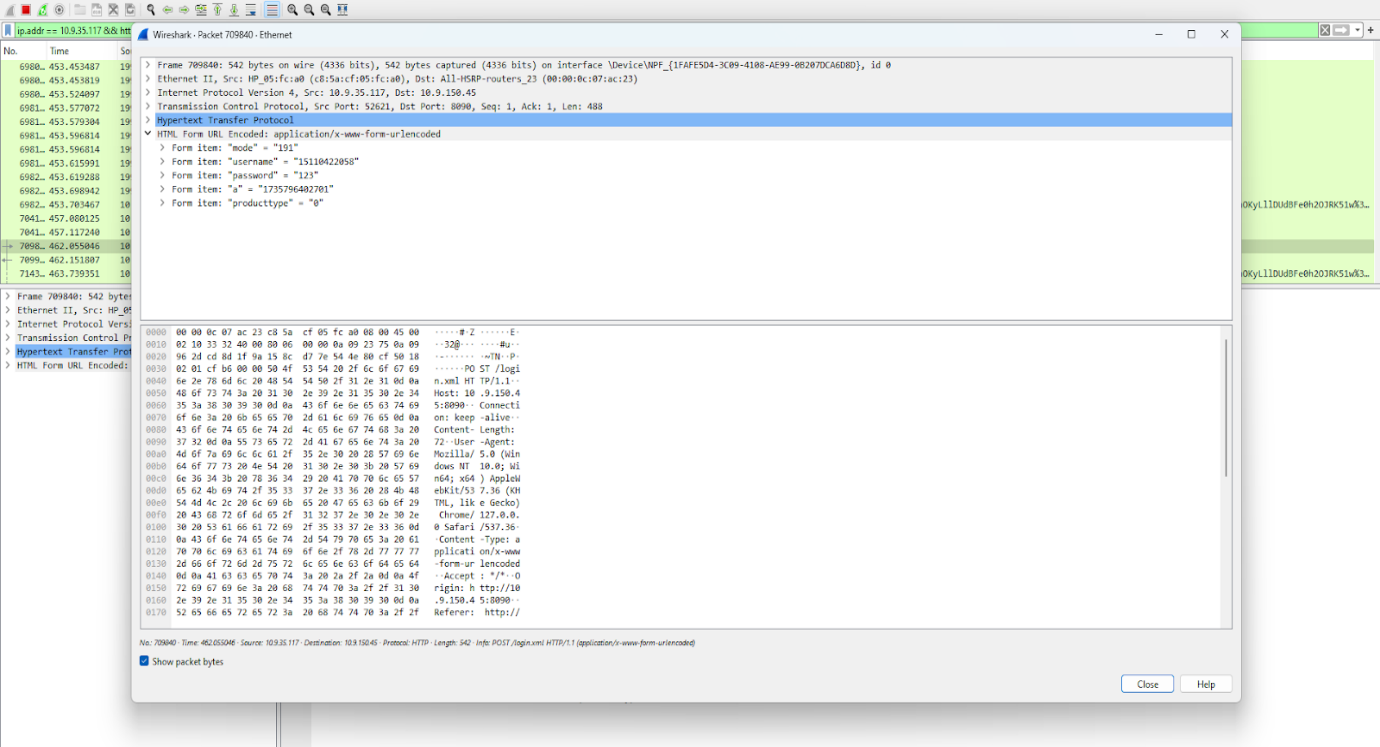
**Step 8: - now in the atmiya login page enter the false username and password.**



**Step 9: - now just double click on the login.xml to view its details**



**Step 10: - now you will see the false details in this interface.**



1. **WAP to find the factorial of the given number**

**🡺 Code: -**

# Function to calculate factorial

def factorial(n):

    if n == 0 or n == 1:

        return 1

    else:

        return n \* factorial(n - 1)

# Input from the user

num = int(input("Enter a number: "))

# Check if the number is non-negative

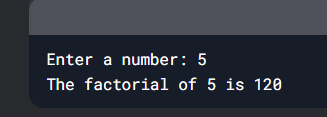
if num < 0:

    print("Factorial is not defined for negative numbers.")

else:

    print(f"The factorial of {num} is {factorial(num)}")

**🡺 Output: -**

****

1. **WAP to find the given number is armstrong or not.?**

**🡺Code: -**

# Function to check if a number is an Armstrong number

def is\_armstrong(num):

    # Convert the number to a string to find the number of digits

    num\_str = str(num)

    length = len(num\_str)

    # Calculate the sum of each digit raised to the power of the number of digits

    sum\_of\_digits = sum(int(digit) \*\* length for digit in num\_str)

    # Check if the sum is equal to the original number

    return sum\_of\_digits == num

# Input from the user

number = int(input("Enter a number: "))

# Check if the number is an Armstrong number

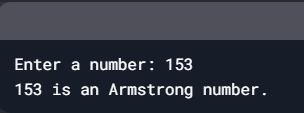
if is\_armstrong(number):

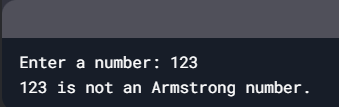
    print(f"{number} is an Armstrong number.")

else:

    print(f"{number} is not an Armstrong number.")

**🡺Output: -**

****

****

1. **WAP to find if a given number is palindrome or not.?**

**🡺Code: -**

# Function to check if a number is a palindrome

def is\_palindrome(num):

    # Convert the number to a string

    num\_str = str(num)

    # Check if the string is equal to its reverse

    return num\_str == num\_str[::-1]

# Input from the user

number = int(input("Enter a number: "))

# Check if the number is a palindrome

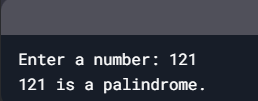
if is\_palindrome(number):

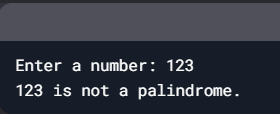
    print(f"{number} is a palindrome.")

else:

    print(f"{number} is not a palindrome.")

**🡺Output: -**

****

****

1. **WAP to find the given number is prime or Not.**

**🡺Code: -**

# Function to check if a number is prime

def is\_prime(num):

    # Numbers less than or equal to 1 are not prime

    if num <= 1:

        return False

    # Check for factors from 2 to the square root of the number

    for i in range(2, int(num\*\*0.5) + 1):

        if num % i == 0:

            return False

    return True

# Input from the user

number = int(input("Enter a number: "))

# Check if the number is prime

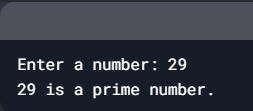
if is\_prime(number):

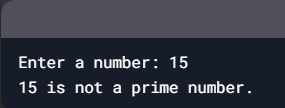
    print(f"{number} is a prime number.")

else:

    print(f"{number} is not a prime number.")

**🡺Output: -**

****

****

1. **WAP to find the prime number from the given range by user**

**🡺Code: -**

# Function to check if a number is prime

def is\_prime(num):

    if num <= 1:

        return False

    for i in range(2, int(num\*\*0.5) + 1):

        if num % i == 0:

            return False

    return True

# Input from the user

start = int(input("Enter the start of the range: "))

end = int(input("Enter the end of the range: "))

# Find and print prime numbers in the given range

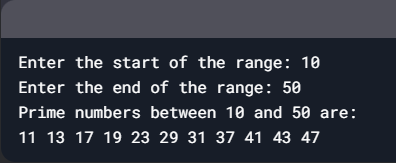
print(f"Prime numbers between {start} and {end} are:")

for number in range(start, end + 1):

    if is\_prime(number):

        print(number, end=" ")

**🡺Output: -**

****

1. **WAP to find the factors of given number (ex - 6 -> 1,2,3,6)**

**🡺Code: -**

# Function to find factors of a number

def find\_factors(num):

    factors = []

    # Iterate from 1 to the number itself

    for i in range(1, num + 1):

        if num % i == 0:  # Check if 'i' is a factor

            factors.append(i)

    return factors

# Input from the user

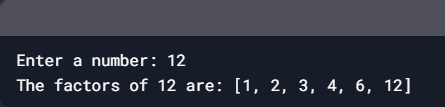
number = int(input("Enter a number: "))

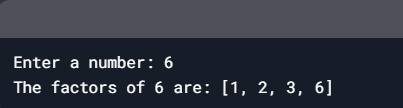
# Find and print the factors

factors = find\_factors(number)

print(f"The factors of {number} are: {factors}")

**🡺Output: -**

****

****

1. **WAP to print given number into word (ex - 25 -> twenty-five)**

**🡺 Code: -**

# Function to convert a number into words

def number\_to\_words(num):

    # Lists for words

    units = ["", "one", "two", "three", "four", "five", "six", "seven", "eight", "nine"]

    teens = ["ten", "eleven", "twelve", "thirteen", "fourteen", "fifteen", "sixteen",

             "seventeen", "eighteen", "nineteen"]

    tens = ["", "ten", "twenty", "thirty", "forty", "fifty", "sixty", "seventy",

            "eighty", "ninety"]

    if num == 0:

        return "zero"

    elif num < 10:

        return units[num]

    elif 10 <= num < 20:

        return teens[num - 10]

    elif 20 <= num < 100:

        return tens[num // 10] + (" " + units[num % 10] if num % 10 != 0 else "")

    elif 100 <= num < 1000:

        return units[num // 100] + " hundred" + (" " + number\_to\_words(num % 100) if num % 100 != 0 else "")

    else:

        return "Number out of range (0-999)"

# Input from the user

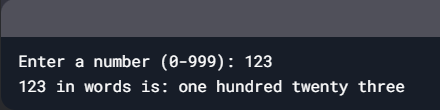
number = int(input("Enter a number (0-999): "))

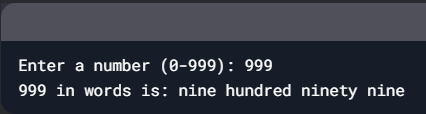
# Convert and print the number in words

words = number\_to\_words(number)

print(f"{number} in words is: {words}")

**🡺Output: -**

****

****

1. Write a C Program to Print the Multiplication Table of given number

🡺**Code: -**

# Input from the user

number = int(input("Enter a number: "))

limit = int(input("Enter the limit for the table: "))

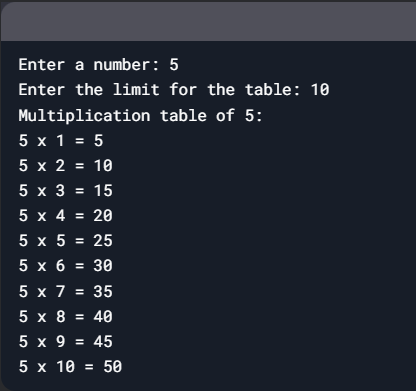
# Print the multiplication table

print(f"Multiplication table of {number}:")

for i in range(1, limit + 1):

    print(f"{number} x {i} = {number \* i}")

**🡺 Output: -**

****

1. [**Write a Program to Reverse a Number Using any Loop**](https://www.programmingwithbasics.com/2023/02/program-to-reverse-number.html)

**🡺Code: -**

# Function to reverse a number

def reverse\_number(num):

    reversed\_num = 0

    while num > 0:

        # Extract the last digit

        digit = num % 10

        # Append the digit to the reversed number

        reversed\_num = reversed\_num \* 10 + digit

        # Remove the last digit from the original number

        num = num // 10

    return reversed\_num

# Input from the user

number = int(input("Enter a number: "))

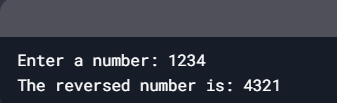
# Reverse the number

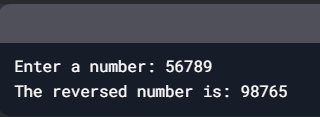
reversed\_number = reverse\_number(number)

# Output the result

print(f"The reversed number is: {reversed\_number}")

**🡺Output: -**

****

****

1. **Write a Program to** [**Count the Number of Digits of an Integer**](https://www.programmingwithbasics.com/2016/03/c-program-to-count-number-of-digits-of.html)

**🡺Code: -**

def count\_digits(num):

    num = abs(num)

    if num == 0:

        return 1  # 0 has 1 digit

    count = 0

    while num > 0:

        count += 1

        num = num // 10

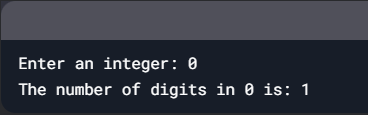
    return count

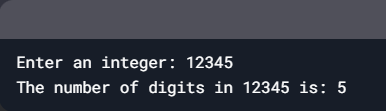
number = int(input("Enter an integer: "))

digit\_count = count\_digits(number)

print(f"The number of digits in {number} is: {digit\_count}")

**🡺Output: -**

****

****

1. **Write a Program to find the sum of two Matrix**

**🡺Code: -**

# Function to add two matrices

def add\_matrices(matrix1, matrix2):

    # Check if the matrices have the same dimensions

    if len(matrix1) != len(matrix2) or len(matrix1[0]) != len(matrix2[0]):

        return "Matrices must have the same dimensions for addition."

    # Initialize a result matrix with zeros

    result = [[0 for \_ in range(len(matrix1[0]))] for \_ in range(len(matrix1))]

    # Perform addition

    for i in range(len(matrix1)):

        for j in range(len(matrix1[0])):

            result[i][j] = matrix1[i][j] + matrix2[i][j]

    return result

# Input matrices

matrix1 = [

    [1, 2, 3],

    [4, 5, 6],

    [7, 8, 9]

]

matrix2 = [

    [9, 8, 7],

    [6, 5, 4],

    [3, 2, 1]

]

# Add the matrices

result = add\_matrices(matrix1, matrix2)

# Output the result

if isinstance(result, str):

    print(result)

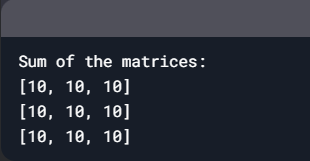
else:

    print("Sum of the matrices:")

    for row in result:

        print(row)

**🡺Output: -**

****

1. **Write a Program to find the Multiplication of two Matrix**

🡺**Code: -**

# Function to multiply two matrices

def multiply\_matrices(matrix1, matrix2):

    # Check if the matrices can be multiplied

    if len(matrix1[0]) != len(matrix2):

        return "Matrices cannot be multiplied. Number of columns in the first matrix must equal the number of rows in the second matrix."

    # Initialize a result matrix with zeros

    result = [[0 for \_ in range(len(matrix2[0]))] for \_ in range(len(matrix1))]

    # Perform multiplication

    for i in range(len(matrix1)):  # Rows of matrix1

        for j in range(len(matrix2[0])):  # Columns of matrix2

            for k in range(len(matrix2)):  # Rows of matrix2

                result[i][j] += matrix1[i][k] \* matrix2[k][j]

    return result

# Input matrices

matrix1 = [

    [1, 2, 3],

    [4, 5, 6]

]

matrix2 = [

    [7, 8],

    [9, 10],

    [11, 12]

]

# Multiply the matrices

result = multiply\_matrices(matrix1, matrix2)

# Output the result

if isinstance(result, str):

    print(result)

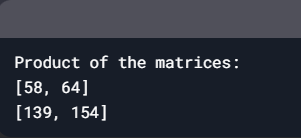
else:

    print("Product of the matrices:")

    for row in result:

        print(row)

**🡺Output: - Ss**

****

1. **Write a Program to remove all the Vowels from the String**

**🡺Code: -**

# Function to remove vowels from a string

def remove\_vowels(input\_string):

    # Define vowels

    vowels = "aeiouAEIOU"

    # Use list comprehension to filter out vowels

    result\_string = ''.join([char for char in input\_string if char not in vowels])

    return result\_string

# Input from the user

input\_string = input("Enter a string: ")

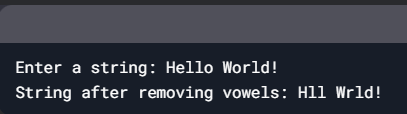
# Remove vowels

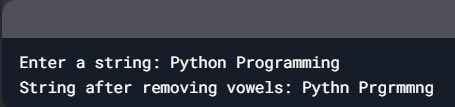
result = remove\_vowels(input\_string)

# Output the result

print("String after removing vowels:", result)

**🡺Output: -**





1. **Write a Program to remove the Duplicate element from the string**

**🡺Code: -**

def remove\_duplicates(input\_string):

    seen = set()

    result = []

    for char in input\_string:

        if char.lower() not in seen:  # Convert to lowercase for case-insensitivity

            seen.add(char.lower())

            result.append(char)

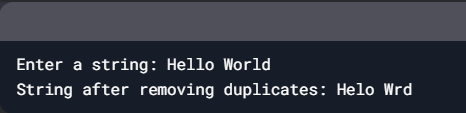
    return ''.join(result)

input\_string = input("Enter a string: ")

result = remove\_duplicates(input\_string)

print("String after removing duplicates:", result)

**🡺Output: -**



1. **Write a Program to check given string in Camel Case or Not.?**

**🡺Code: -**

# Function to check if a string is in Camel Case

def is\_camel\_case(input\_string):

    if not input\_string:  # Check if the string is empty

        return False

    # Check if the first character is lowercase and there are no spaces or underscores

    if input\_string[0].isupper() or ' ' in input\_string or '\_' in input\_string:

        return False

    # Check if there is at least one uppercase letter (Camel Case requirement)

    return any(char.isupper() for char in input\_string)

# Input from the user

input\_string = input("Enter a string: ")

# Check if the string is in Camel Case

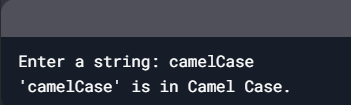
if is\_camel\_case(input\_string):

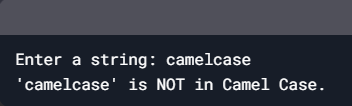
    print(f"'{input\_string}' is in Camel Case.")

else:

    print(f"'{input\_string}' is NOT in Camel Case.")

**🡺Output: -**





1. **WAP to find indices of array – Input = [2,2,3] sum = 5 then O/P = [1,2] indices**

**🡺Code: -**

# Function to find indices of two numbers that add up to the target sum

def find\_indices(nums, target):

    # Create a dictionary to store the difference and its index

    num\_to\_index = {}

    for index, num in enumerate(nums):

        # Calculate the difference needed to reach the target

        difference = target - num

        # Check if the difference is already in the dictionary

        if difference in num\_to\_index:

            return [num\_to\_index[difference], index]

        # Store the current number and its index in the dictionary

        num\_to\_index[num] = index

    return None  # Return None if no such pair is found

# Input array and target sum

nums = [2, 2, 3]

target = 5

# Find the indices

result = find\_indices(nums, target)

# Output the result

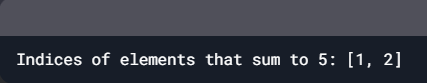
if result:

    print(f"Indices of elements that sum to {target}: {result}")

else:

    print("No such pair found.")

**🡺Output: -**



1. **WAP to convert Roman to Integer Number**

**🡺Code: -**

# Function to convert Roman numeral to integer

def roman\_to\_int(roman):

    # Dictionary to map Roman numerals to their integer values

    roman\_to\_int\_map = {

        'I': 1,

        'V': 5,

        'X': 10,

        'L': 50,

        'C': 100,

        'D': 500,

        'M': 1000

    }

    total = 0

    prev\_value = 0

    # Iterate through the Roman numeral string from right to left

    for char in reversed(roman):

        current\_value = roman\_to\_int\_map[char]

        # If the current value is less than the previous value, subtract it

        if current\_value < prev\_value:

            total -= current\_value

        else:

            total += current\_value

        prev\_value = current\_value

    return total

# Input from the user

roman\_numeral = input("Enter a Roman numeral: ").upper()

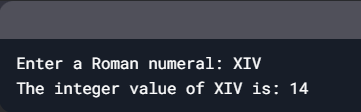
# Convert Roman numeral to integer

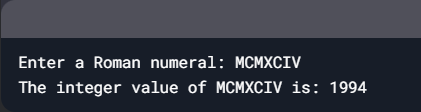
integer\_value = roman\_to\_int(roman\_numeral)

# Output the result

print(f"The integer value of {roman\_numeral} is: {integer\_value}")

**🡺Output: -**





1. **WAP to find Longest common prefix Input = [“Flower”,”Flow”,”Flight”]  O/P = “Fl”**

**🡺Code: -**

# Function to find the longest common prefix

def longest\_common\_prefix(strs):

    if not strs:  # Check if the input list is empty

        return ""

    # Find the shortest string in the list

    shortest\_str = min(strs, key=len)

    # Iterate through the characters of the shortest string

    for i, char in enumerate(shortest\_str):

        # Compare the character with the corresponding character in other strings

        for other\_str in strs:

            if other\_str[i] != char:

                return shortest\_str[:i]  # Return the prefix up to the mismatch

    return shortest\_str  # If no mismatch, return the entire shortest string

# Input list of strings

input\_strings = ["Flower", "Flow", "Flight"]

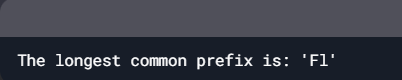
# Find the longest common prefix

result = longest\_common\_prefix(input\_strings)

# Output the result

print(f"The longest common prefix is: '{result}'")

**🡺Output: -**



1. **WAP to find permutation of given number I/P – [1,2,3] to all possibilities**

**🡺Code: -**

# Function to find all permutations recursively

def find\_permutations(input\_list):

    if len(input\_list) == 0:

        return []

    if len(input\_list) == 1:

        return [input\_list]

    # List to store all permutations

    result = []

    for i in range(len(input\_list)):

        # Extract the current element

        current\_element = input\_list[i]

        # Remaining list after removing the current element

        remaining\_list = input\_list[:i] + input\_list[i+1:]

        # Generate all permutations of the remaining list

        for perm in find\_permutations(remaining\_list):

            result.append([current\_element] + perm)

    return result

# Input list

input\_list = [1, 2, 3]

# Find all permutations

permutation\_list = find\_permutations(input\_list)

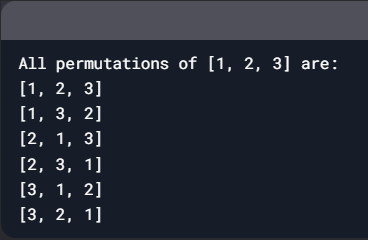
# Output the result

print("All permutations of", input\_list, "are:")

for perm in permutation\_list:

    print(perm)

**🡺Output: -**



1. **WAP to count and say I/P – 332251 -> two 3’s, two 2’s, one 5 one 1’s**

**🡺Code: -**

# Function to perform the "Count and Say" operation

def count\_and\_say(number):

    # Convert the number to a string for easy iteration

    number\_str = str(number)

    result = ""

    i = 0

    n = len(number\_str)

    while i < n:

        # Get the current digit

        current\_digit = number\_str[i]

        count = 1

        # Count the number of times the current digit repeats

        while i + 1 < n and number\_str[i + 1] == current\_digit:

            count += 1

            i += 1

        # Append the count and the digit to the result

        result += f"{count} {current\_digit}{'s' if count > 1 else ''}, "

        i += 1

    # Remove the trailing comma and space

    return result.rstrip(", ")

# Input number

input\_number = 332251

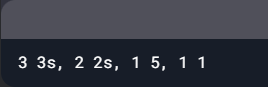
# Perform the "Count and Say" operation

output = count\_and\_say(input\_number)

# Output the result

print(output)

**🡺Output: -**



1. **WAP to rotate 3\*3 Matrix from row to column**

**🡺Code: -**

# Function to rotate a 3x3 matrix (transpose)

def rotate\_matrix(matrix):

    # Use list comprehension to transpose the matrix

    return [[matrix[j][i] for j in range(len(matrix))] for i in range(len(matrix[0]))]

# Input 3x3 matrix

matrix = [

    [1, 2, 3],

    [4, 5, 6],

    [7, 8, 9]

]

# Rotate the matrix

rotated\_matrix = rotate\_matrix(matrix)

# Output the original and rotated matrix

print("Original Matrix:")

for row in matrix:

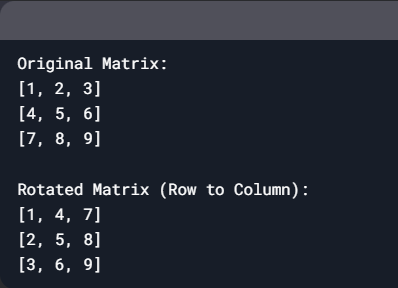
    print(row)

print("\nRotated Matrix (Row to Column):")

for row in rotated\_matrix:

    print(row)

**🡺Output: -**



1. **WAP to draw Square Hollow pattern**

**🡺Code: -**

# Function to draw a square hollow pattern

def draw\_square\_hollow(n):

    for i in range(n):  # Loop for rows

        for j in range(n):  # Loop for columns

            # Print '\*' for the first and last row, or first and last column

            if i == 0 or i == n - 1 or j == 0 or j == n - 1:

                print("\*", end=" ")

            else:

                print(" ", end=" ")  # Print space for the hollow part

        print()  # Move to the next line after each row

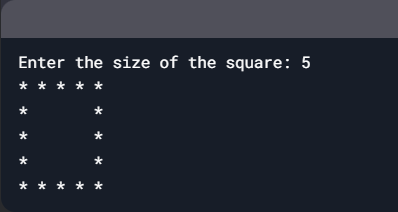
# Input size of the square

size = int(input("Enter the size of the square: "))

# Draw the square hollow pattern

draw\_square\_hollow(size)

**🡺Output: -**



1. **WAP to print the Reverse pattern.**

**1**

**2 2**

**3 3 3**

**🡺Code: -**

# Function to print the reverse pattern

def print\_reverse\_pattern(n):

    for i in range(1, n + 1):  # Loop for rows

        # Print spaces for alignment

        for j in range(n - i):

            print(" ", end=" ")

        # Print the numbers

        for k in range(i):

            print(i, end=" ")

        print()  # Move to the next line after each row

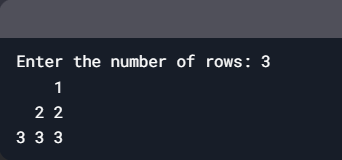
# Input number of rows

rows = int(input("Enter the number of rows: "))

# Print the reverse pattern

print\_reverse\_pattern(rows)

**🡺Output: -**



1. **WAP for given Pattern.**

**1 2 3 4 5**

**1 2 3 4**

**1 2 3**

**1 2**

**1**

**🡺Code: -**

# Function to print the pattern

def print\_pattern(n):

    for i in range(n, 0, -1):  # Loop for rows in reverse order

        # Print numbers from 1 to i

        for j in range(1, i + 1):

            print(j, end=" ")

        print()  # Move to the next line after each row

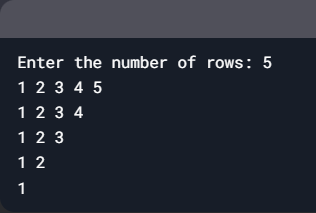
# Input number of rows

rows = int(input("Enter the number of rows: "))

# Print the pattern

print\_pattern(rows)

**🡺Output: -**



1. **WAP for given Triangle Pattern.**

**1**

**2 3**

**4 5 6**

**7 8 9 10**

**🡺Code: -**

# Function to print the triangle pattern with spacing

def print\_triangle\_pattern(n):

    num = 1  # Initialize the starting number

    for i in range(1, n + 1):  # Loop for rows

        # Print spaces for alignment

        for j in range(n - i):

            print(" ", end="  ")

        # Print numbers with spacing

        for k in range(i):

            print(num, end="   ")  # Add extra spaces between numbers

            num += 1  # Increment the number

        print()  # Move to the next line after each row

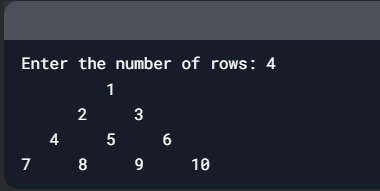
# Input number of rows

rows = int(input("Enter the number of rows: "))

# Print the triangle pattern

print\_triangle\_pattern(rows)

**🡺Output: -**



1. **WAP to print the given pattern.**

**1**

**1 2**

**1 2 3**

**🡺Code: -**

# Function to print the pattern

def print\_pattern(n):

    for i in range(1, n + 1):  # Loop for rows

        for j in range(1, i + 1):  # Loop for columns

            print(j, end=" ")  # Print numbers from 1 to i

        print()  # Move to the next line after each row

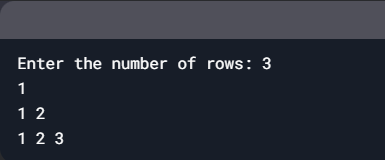
# Input number of rows

rows = int(input("Enter the number of rows: "))

# Print the pattern

print\_pattern(rows)

**🡺Output: -**



1. **WAP to print the given pattern.**

**\***

**\*\*\***

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

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

**🡺Code: -**

# Function to print the pyramid pattern

def print\_pyramid(n):

    for i in range(1, n + 1):  # Loop for rows

        # Print spaces for alignment

        for j in range(n - i):

            print(" ", end=" ")

        # Print asterisks

        for k in range(2 \* i - 1):

            print("\*", end=" ")

        print()  # Move to the next line after each row

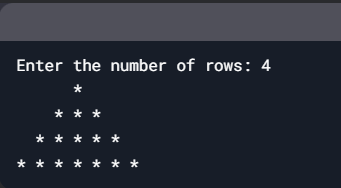
# Input number of rows

rows = int(input("Enter the number of rows: "))

# Print the pyramid pattern

print\_pyramid(rows)

**🡺Output: -**



1. **WAP to print the given pattern.**

**\***

**\* \***

**\*   \***

**\*     \***

**\*       \***

**\*         \***

**\*       \***

**\*     \***

**\*   \***

**\* \***

**\***

**🡺Code: -**

# Function to print the hollow diamond pattern

def print\_hollow\_diamond(n):

    # Upper part of the diamond

    for i in range(1, n + 1):

        # Print spaces for alignment

        for j in range(n - i):

            print(" ", end="")

        # Print asterisks and spaces

        for k in range(2 \* i - 1):

            if k == 0 or k == 2 \* i - 2:

                print("\*", end="")

            else:

                print(" ", end="")

        print()  # Move to the next line after each row

    # Lower part of the diamond

    for i in range(n - 1, 0, -1):

        # Print spaces for alignment

        for j in range(n - i):

            print(" ", end="")

        # Print asterisks and spaces

        for k in range(2 \* i - 1):

            if k == 0 or k == 2 \* i - 2:

                print("\*", end="")

            else:

                print(" ", end="")

        print()  # Move to the next line after each row

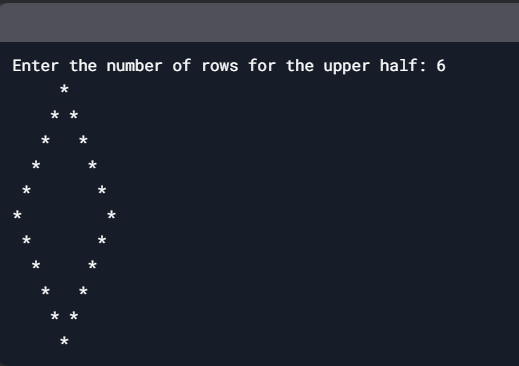
# Input number of rows (for the upper half)

rows = int(input("Enter the number of rows for the upper half: "))

# Print the hollow diamond pattern

print\_hollow\_diamond(rows)

**🡺Output: -**



1. **WAP to print Right Pascal’s Triangle pattern**

**🡺Code: -**

# Function to print the Right Pascal's Triangle

def print\_right\_pascal(n):

    # Upper part of the triangle

    for i in range(1, n + 1):

        for j in range(i):

            print("\*", end=" ")

        print()  # Move to the next line after each row

    # Lower part of the triangle

    for i in range(n - 1, 0, -1):

        for j in range(i):

            print("\*", end=" ")

        print()  # Move to the next line after each row

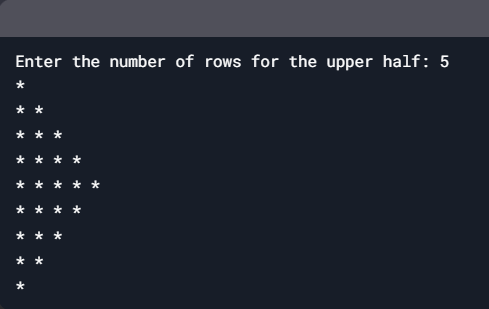
# Input number of rows for the upper half

rows = int(input("Enter the number of rows for the upper half: "))

# Print the Right Pascal's Triangle

print\_right\_pascal(rows)

**🡺Output: -**



1. **WAP to print Spiral Matrix output, starting from left to right row wise.**

**🡺Code: -**

# Function to print a spiral matrix

def print\_spiral\_matrix(n):

    # Initialize a n x n matrix with zeros

    matrix = [[0 for \_ in range(n)] for \_ in range(n)]

    # Define starting positions and boundaries

    left, right = 0, n - 1

    top, bottom = 0, n - 1

    num = 1  # Starting number

    while left <= right and top <= bottom:

        # Traverse from left to right (top row)

        for i in range(left, right + 1):

            matrix[top][i] = num

            num += 1

        top += 1

        # Traverse from top to bottom (right column)

        for i in range(top, bottom + 1):

            matrix[i][right] = num

            num += 1

        right -= 1

        # Traverse from right to left (bottom row)

        if top <= bottom:

            for i in range(right, left - 1, -1):

                matrix[bottom][i] = num

                num += 1

            bottom -= 1

        # Traverse from bottom to top (left column)

        if left <= right:

            for i in range(bottom, top - 1, -1):

                matrix[i][left] = num

                num += 1

            left += 1

    # Print the spiral matrix

    for row in matrix:

        print(" ".join(map(str, row)))

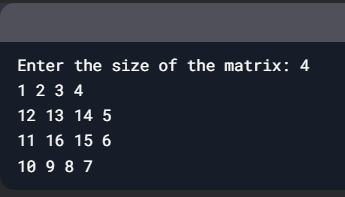
# Input size of the matrix

size = int(input("Enter the size of the matrix: "))

# Print the spiral matrix

print\_spiral\_matrix(size)

**🡺Output: -**



1. **WAP to remove duplicate elements from the given elements**

**🡺Code: -**

# Function to remove duplicate elements

def remove\_duplicates(elements):

    # Use a set to store unique elements while preserving order

    unique\_elements = []

    seen = set()

    for element in elements:

        if element not in seen:

            seen.add(element)

            unique\_elements.append(element)

    return unique\_elements

# Input list with duplicate elements

elements = input("Enter elements (comma-separated): ").split(",")

elements = [item.strip() for item in elements]  # Remove extra spaces

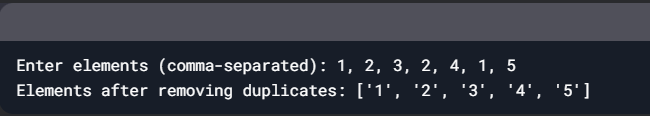
# Remove duplicates

result = remove\_duplicates(elements)

# Output the result

print("Elements after removing duplicates:", result)

**🡺Output: -**



1. **WAP to check whether a given string is isomorphic or not.?**

**🡺Code: -**

# Function to check if two strings are isomorphic

def is\_isomorphic(s, t):

    if len(s) != len(t):  # If lengths are different, they cannot be isomorphic

        return False

    # Create two dictionaries to map characters from s to t and vice versa

    s\_to\_t = {}

    t\_to\_s = {}

    for i in range(len(s)):

        char\_s = s[i]

        char\_t = t[i]

        # Check if the mapping from s to t is consistent

        if char\_s in s\_to\_t:

            if s\_to\_t[char\_s] != char\_t:

                return False

        else:

            s\_to\_t[char\_s] = char\_t

        # Check if the mapping from t to s is consistent

        if char\_t in t\_to\_s:

            if t\_to\_s[char\_t] != char\_s:

                return False

        else:

            t\_to\_s[char\_t] = char\_s

    return True

# Input two strings

s = input("Enter the first string: ")

t = input("Enter the second string: ")

# Check if the strings are isomorphic

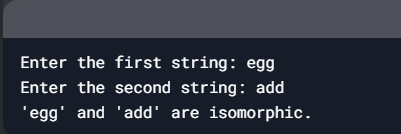
if is\_isomorphic(s, t):

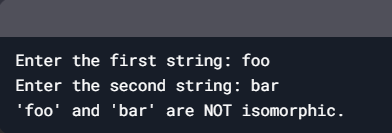
    print(f"'{s}' and '{t}' are isomorphic.")

else:

    print(f"'{s}' and '{t}' are NOT isomorphic.")

🡺**Output: -**





1. **WAP to Replace a character at a specific index in a String**

**🡺Code: -**

# Function to replace a character at a specific index

def replace\_char(input\_string, index, new\_char):

    # Check if the index is valid

    if index < 0 or index >= len(input\_string):

        return "Invalid index"

    # Convert the string to a list (strings are immutable in Python)

    string\_list = list(input\_string)

    # Replace the character at the specified index

    string\_list[index] = new\_char

    # Convert the list back to a string

    return "".join(string\_list)

# Input from the user

input\_string = input("Enter a string: ")

index = int(input("Enter the index to replace: "))

new\_char = input("Enter the new character: ")

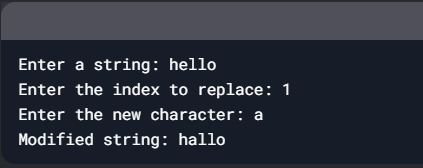
# Replace the character

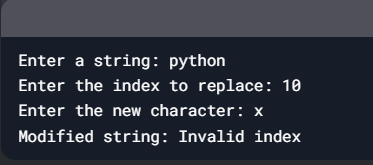
result = replace\_char(input\_string, index, new\_char)

# Output the result

print("Modified string:", result)

🡺**Output: -**





1. **WAP to Swap Two Strings without Third String Variable**

**🡺Code: -**

# Input two strings

string1 = input("Enter the first string: ")

string2 = input("Enter the second string: ")

# Swap the strings without using a third variable

string1, string2 = string2, string1

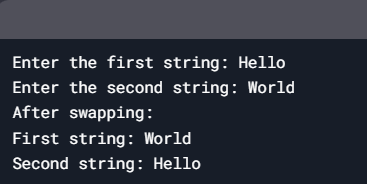
# Output the swapped strings

print("After swapping:")

print("First string:", string1)

print("Second string:", string2)

**🡺Output: -**



1. **WAP to Count Number of Words in String**

**🡺Code: -**

# Function to count the number of words in a string

def count\_words(input\_string):

    # Split the string into words using spaces as the delimiter

    words = input\_string.split()

    # Return the number of words

    return len(words)

# Input from the user

input\_string = input("Enter a string: ")

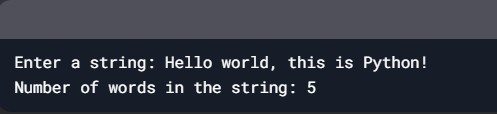
# Count the number of words

word\_count = count\_words(input\_string)

# Output the result

print("Number of words in the string:", word\_count)

**🡺Output: -**



1. **WAP to find specific word from given string**

**🡺Code: -**

# Function to find a specific word in a string

def find\_word(input\_string, word):

    # Split the string into words

    words = input\_string.split()

    # Check if the word exists in the list of words

    if word in words:

        return f"The word '{word}' was found in the string."

    else:

        return f"The word '{word}' was NOT found in the string."

# Input from the user

input\_string = input("Enter a string: ")

word\_to\_find = input("Enter the word to find: ")

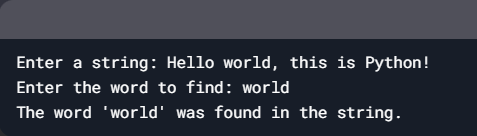
# Find the word

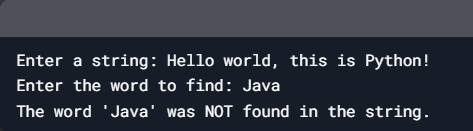
result = find\_word(input\_string, word\_to\_find)

# Output the result

print(result)

**🡺Output: -**





1. **WAP to find the GCD of the inputted 2 numbers**

**🡺Code: -**

# Function to find GCD using the Euclidean Algorithm

def find\_gcd(a, b):

    while b != 0:

        a, b = b, a % b  # Update a to b and b to the remainder of a divided by b

    return a

# Input two numbers from the user

num1 = int(input("Enter the first number: "))

num2 = int(input("Enter the second number: "))

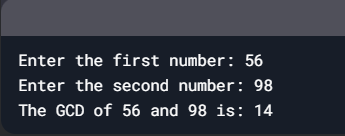
# Find the GCD

gcd = find\_gcd(num1, num2)

# Output the result

print(f"The GCD of {num1} and {num2} is: {gcd}")

**🡺Output: -**



1. **WAP to sort elements in given Array**

**🡺Code: -**

# Function to sort an array in place

def sort\_array(arr):

    arr.sort()  # Sort the array in place

# Input array from the user

input\_array = input("Enter elements of the array (comma-separated): ").split(",")

input\_array = [int(item.strip()) for item in input\_array]  # Convert to integers

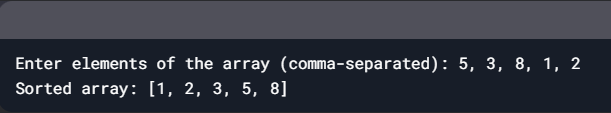
# Sort the array in place

sort\_array(input\_array)

# Output the sorted array

print("Sorted array:", input\_array)

**🡺Output: -**



1. **WAP to find the Reverse of the Array**

**🡺Code: -**

# Function to reverse an array in place

def reverse\_array(arr):

    arr.reverse()  # Reverse the array in place

# Input array from the user

input\_array = input("Enter elements of the array (comma-separated): ").split(",")

input\_array = [item.strip() for item in input\_array]  # Remove extra spaces

# Reverse the array in place

reverse\_array(input\_array)

# Output the reversed array

print("Reversed array:", input\_array)

**🡺Output: -**

