Input-Output

Scenario 1:

Display Product Information
You need to create a program that accepts details about a product (name, price, and availability) and displays the information in a structured format. This question evaluates your understanding of Python's basic syntax and input/output operations.

Input Format:

- Name of the product (string).
- Price of the product (float).
- Availability (boolean).

Constraints:

- Product name length ≤ 50 characters.
- Price ≥ 0.
- Availability should be True or False.

Output Format:

- Three lines displaying product details.

Sample Inputs and Outputs:

Sample Input	Sample Output
Laptop 59999.99 True	Product Name: Laptop Price: 59999.99 In Stock: True
Phone 29999.50 False	Product Name: Phone Price: 29999.50 In Stock: False
Tablet 15000.0 True	Product Name: Tablet Price: 15000.0 In Stock: True
Desktop 45000.75 False	Product Name: Desktop Price: 45000.75 In Stock: False
Camera 25000.0 True	Product Name: Camera

	Price: 25000.0 In Stock: True

Scenario 2:

Check Variable Type

In Python, variables are dynamically typed, meaning their type is determined at runtime based on the value assigned. Common types include integers (int), floating-point numbers (float), and strings (str). You can check a variable's type using the type() function.

This task will identify the type of a given input value.

Input Format:

- A single variable and its value.

Constraints:

- The variable can only be of types: integer, string, or float.

Output Format:

- The type of the variable (e.g., int, str, or float).

Sample Inputs and Outputs:

Sample Input	Sample Output
42	int
"Hello, Python!"	str
3.14	float
100	int
"123"	str

Scenario 3:

Simple Calculator

This program is a basic calculator that can perform addition, subtraction, multiplication, or division based on user input. It ensures that division by zero is handled appropriately. This question tests your understanding of operators and control flow in Python.

Input Format:

- Two space-separated floats (numbers).
- A single character representing an operator (+, -, *, /).

Constraints:

- Second number $\neq 0$ for division.

Output Format:

- A single line displaying the result. If division by zero is attempted, display "Error: Division by zero".

Sample Inputs and Outputs:

Sample Input	**Sample Output**
10 5 +	15.0
8 4 -	4.0
6 3 *	18.0
12 4 /	3.0
5 0 /	Error: Division by zero

Scenario 4:

Print Multiplication Table

A multiplication table is a mathematical table used to define a multiplication operation for a specific integer. In Python, this can be achieved using loops.

This task will print the multiplication table for a given number up to 10 rows.

Input Format:

- An integer nnn, whose table needs to be printed.

Constraints:

- 1≤n≤101 \leq n \leq 101≤n≤10.

Output Format:

- A multiplication table for nnn up to 10 rows.

Sample Inputs and Outputs:

Sample Input	Sample Output
2	2 x 1 = 2 2 x 2 = 4 ...
3	3 x 1 = 3 3 x 2 = 6 ...
5	5 x 1 = 5 5 x 2 = 10 ...
7	7 x 1 = 7 7 x 2 = 14 ...
10	10 x 1 = 10 10 x 2 = 20 ...

Scenario 5:

Check Leap Year

A year is a leap year if it is divisible by 4, but not divisible by 100 unless it is also divisible by 400.

This program determines if a given year is a leap year or not.
Input Format:

- A single integer representing the year.

Constraints:

- 1≤year≤1041 \leq \text{year} \leq 10^41≤year≤104.

Output Format:

- "Leap Year" if the year is a leap year.
- "Not a Leap Year" otherwise.

Sample Inputs and Outputs:

Sample Input	Sample Output
2000	Leap Year
1900	Not a Leap Year
2024	Leap Year
2100	Not a Leap Year
2023	Not a Leap Year

Scenario 6:

Analyze String Input
Write a program that analyzes a string to count the number of vowels, consonants, and digits. This problem helps you practice loops, conditional statements, and string operations in Python.

Input Format:

- A single line containing a string.

Constraints:

- String length ≤ 1000 characters.
- String can contain alphabets, digits, and special characters.

Output Format:

- Three lines displaying:
    1. Number of vowels.
    2. Number of consonants.
    3. Number of digits.

Sample Inputs and Outputs:

Sample Input	**Sample Output**
Hello123	Number of Vowels: 2 Number of Consonants: 3 Number of Digits: 3
Python123	Number of Vowels: 1 Number of Consonants: 5 Number of Digits: 3
ProgrammingIsFun456	Number of Vowels: 5 Number of Consonants: 12 Number of Digits: 3
Data123Science	Number of Vowels: 5 Number of Consonants: 7 Number of Digits: 3
12345	Number of Vowels: 0 Number of Consonants: 0 Number of Digits: 5

Scenario 7:

Create a Calculator

Python supports various arithmetic operations, such as addition (+), subtraction (-), multiplication (*), and division (/). This task involves evaluating basic arithmetic expressions entered by the user.

Input Format:

- A single arithmetic expression (e.g., "10 + 20").

Constraints:

- The expression will only include integers and the operators +, -, *, and /.
- Handle division by zero gracefully.

Output Format:

- The result of the calculation.

Sample Inputs and Outputs:

Sample Input	Sample Output
10 + 20	30
100 - 50	50
5 * 6	30
40 / 8	5.0
15 / 0	Error: Division by zero

Control Structures

Scenario 1: Phone Number

A phone number is a 10-digit number. Write a program to calculate the sum of digits for a given phone number.

Explanation with Sample Input and Output

Example 1:

Sample Input: 9876543210

Sample Output: 45

Explanation: The sum of digits for the phone number 9876543210 is calculated as follows: 9 + 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1 + 0 = 45

Example 2:

Sample Input: 1234567890

Sample Output: 45

Explanation: The sum of digits for the phone number 1234567890 is calculated as follows: 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 0 = 45

Test case 1:

Input

1111111111

Output

10

Test case 2:

Input

5555555555

Output

35

Test case 3:

Input

9999999999

Output

90

Test case 4:

Input:

1231231231

Output:

25

Test case 5:

Input:

9879879879

Output:

72

Scenario 2: Printing Natural Numbers in Reverse Order

Rahul is a computer science student who needs to print natural numbers from N to 1, where N is the input provided by the user.

Example 1:

Sample Input: 5

Sample Output: 5 4 3 2 1

Explanation: The program prints all natural numbers between 5 and 1 in reverse order.

Example 2:

Sample Input: 10

Sample Output: 10 9 8 7 6 5 4 3 2 1

Explanation: The program prints all natural numbers between 10 and 1 in reverse order.

Additional Test Cases:

Test Case 1:

Input: 3

Output: 3 2 1

Test Case 2:

Input: 8

Output: 8 7 6 5 4 3 2 1

Test Case 3:

Input: 15

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

Test Case 4:

Input: 20

Output: 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

Test Case 5:

Input: 25

Output: 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

Scenario 3: Generating Fibonacci Sequence

Rohan is a mathematician who needs to generate the Fibonacci sequence up to a given number. 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.

Sample Test Cases

Test Case 1

Input: 8

Output: [0, 1, 1, 2, 3, 5, 8, 13]

Explanation: The Fibonacci sequence up to 8 is [0, 1, 1, 2, 3, 5, 8, 13).

Test Case 2

Input: 3

Output: [0, 1, 1]

Explanation: The Fibonacci sequence up to 3 is [0, 1, 1].

Additional Test Cases

Test Case 1:

Input: 6

Output: [0, 1, 1, 2, 3, 5]

Test Case 2:

Input: 10

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

Test Case 3:

Input: 12

Output: [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]

Test Case 4:

Input: 15

Output: [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377]

Scenario 4: Identifying Prime Numbers

Sohan is a mathematician who needs to identify prime numbers within a given range. A prime number is a natural number greater than 1 that has no positive divisors other than 1 and itself.

Test Cases:

Test Case 1

Input: 11

Output: True

Explanation: 11 is a prime number because it has no divisors other than 1 and itself.

Test Case 2

Input: 15

Output: False

Explanation: 15 is not a prime number because it has divisors other than 1 and itself (3 and 5).

Additional Test Cases

Test Case 1

Input: 7

Output: True

Test Case 2

Input: 20

Output: False

Test Case 3

Input: 23

Output: True

Test Case 4

Input: 30

Output: False

Test Case 5

Input: 37

Output: True

Scenario 5: Checking Palindrome Numbers

Ramesh is a mathematician who needs to check if a given integer is a palindrome or not. A palindrome is a number that reads the same backward as forward.

Test Cases

Test Case 1

Input: 121

Output: True

Explanation: 121 reads as 121 from left to right and from right to left.

Test Case 2

Input: -121

Output: False

Explanation: From left to right, it reads -121. From right to left, it becomes 121-. Therefore it is not a palindrome.

Additional Test Cases

Test Case 1:

Input: 123

Output: False

Test Case 2:

Input: 1331

Output: True

Test Case 3:

Input: -123

Output: False

Test Case 4:

Input: 12321

Output: True

Test Case 5:

Input: 123456

Output: False

Scenario 6: Diamond Pattern

Rohan wants to print a diamond pattern of stars using Python. The pattern should have a height of n, where n is an odd number.

Test Cases

Test Case 1

Input: 5

Output:

```
-

***

*****

***

-
```

Explanation: The diamond pattern has a height of 5, with the middle row having the maximum number of stars.

Test Case 2

Input: 7

Output:

```
-

***

*****

*******

*****

***
```

```
-
```

Explanation: The diamond pattern has a height of 7, with the middle row having the maximum number of stars.

Additional Test Cases

Test Case 1:

Input: 3

Output:

```
-
***
-
```

Test Case 2:

Input: 9

Output:

```
*
***
*****
*******
*********
*******
*****
***
*
```

Test Case 3:

Input: 11

Output:

```
*
***
*****
*******
*********
***********
*********
*******
*****
***
*
```

Test Case 4:

Input: 13

Output:

```
*
***
*****
*******
*********
***********
*************
***********
*********
*******
*****
***
*
```

Test Case 5:

Input: 15

Output:

```
*
***
*****
*******
*********
***********
*************
***************
*************
***********
*********
*******
*****
***
*
```

Scenario 7: Triangle Classification Game

Ramesh is a game developer who wants to create a triangle classification game. In this game, players will input the lengths of three sides of a triangle, and the game will classify the triangle as equilateral, isosceles, or scalene.

Problem Statement

Write a Python function that takes the lengths of three sides of a triangle as input and returns the classification of the triangle.

Test Cases

Test Case 1

Input: 6, 8, 12

Output: Scalene triangle

Explanation: Since all sides of the triangle are unequal, it is classified as a scalene triangle.

Test Case 2

Input: 8, 7, 8

Output: Isosceles triangle

Explanation: Since at least two sides of the triangle are equal, it is classified as an isosceles triangle.

Additional Test Cases

Test Case 1:

Input: 5, 5, 5

Output: Equilateral triangle

Test Case 2:

Input: 3, 4, 5

Output: Scalene triangle

Test Case 3:

Input: 7, 7, 3

Output: Isosceles triangle

Test Case 4:

Input: 9, 9, 9

Output: Equilateral triangle

Test Case 5:

Input: 1, 2, 3

Output: Scalene triangle

Scenario 8: Secret Code Cracker

As a detective, you have been tasked with cracking a secret code that consists of a series of numbers. Each number represents a letter in the alphabet (1=A, 2=B, ..., 26=Z, 27=a, 28=b, ..., 52=z). Your mission is to write a program that takes the code as input and deciphers it into a message.

Problem Statement

Write a Python program that takes a series of numbers as input and converts them into a message using the standard alphabet-to-number substitution (1=A, 2=B, ..., 26=Z, 27=a, 28=b, ..., 52=z).

Test Cases

Test Case 1

Input: 1 2 3 4 5

Output: ABCDE

Explanation: The input numbers correspond to the letters A, B, C, D, and E, respectively.

Test Case 2

Input: 1 3 5 52 1

Output: ACEzA

Explanation: The input numbers correspond to the letters A, C, E, z, and A, respectively.

Additional Test Cases

Test Case 1:

Input: 14 15 20 2 9 19 20 18 1 20 5

Output: NOTBISTRATE

Test Case 2:

Input: 1 18 5 12 12 15

Output: RLOOLL

Test Case 3:

Input: 2 1 20 8 5 19

Output: BARTIE

Test Case 4:

Input: 20 8 5 19 7 15 14 7

Output: HETIGNO

Test Case 5:

Input: 4 9 22 5 18 15 14 20

Output: DJWSEONG

Scenario 9: Leap Year Detector

Ramesh is a calendar enthusiast who wants to create a program to determine whether a given year is a leap year or not. A leap year is either divisible by 4 but not by 100, or divisible by 400.

Problem Statement

Write a Python program to determine if a given year is a leap year or not. The program should take a year as input and print whether it is a leap year or not.

Test Cases

Test Case 1:

Input: 2004

Output: LY

Explanation: 2004 is a leap year because it is divisible by 4.

Test Case 2:

Input: 1900

Output: NLY

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

Additional Test Cases

Test Case 1:

Input: 2020

Output: LY

Test Case 2:

Input: 2000

Output: LY

Test Case 3:

Input: 1999

Output: NLY

Test Case 4:

Input: 2400

Output: LY

Test Case 5:

Input: 1800

Output: NLY

String

1) Prem's Pattern Pursuit:

In the scenic valleys of Uttarakhand, amidst the tranquility of the Himalayas, lives Prem—a talented programmer with a knack for discovering hidden patterns. One calm afternoon, as the sunlight filters through the trees, Prem encounters an intriguing challenge.

The task is to count how many times a given substring appears in a string, including overlaps. This problem requires a keen eye for detail and a deep understanding of string manipulation.

Prem sets up his laptop by the window, where the gentle breeze and the serene surroundings create an ideal environment for coding. His goal is to write a Python program that will accurately count the occurrences of the substring within the string.

As Prem begins to code, he visualizes the string as a tapestry and the substring as a recurring motif. Each occurrence contributes to the overall pattern, revealing the hidden beauty within the text. His methodical approach and keen observation skills ensure that he captures every overlap.

Join Prem on his journey to uncover hidden patterns in strings. Watch as he skillfully navigates the intricacies of string manipulation, transforming a simple task into a captivating exploration. Let's delve into the world of substring occurrences with Prem in the picturesque setting of Uttarakhand.

Input Format:

- A string text. $(1 \le |text| \le 1000)$ and A string sub. $(1 \le |sub| \le 100)$, space-separated

Output Format:

- An integer representing the number of occurrences of sub in text.

Constraints:

- Both text and sub contain only lowercase letters.

Example:

Example 1:

Input:
"abababa" "aba"

Output:
3

Explanation:
The substring "aba" appears at positions 1, 3, and 5.

Example 2:

Input:

"aaaa" "aa"

Output:
3

Explanation:
The substring "aa" appears at positions 1, 2, and 3.

Test Cases:

Test Case 1:

Input: "mississippi" "iss"

Output: 2

Test Cases 2:

Input: "aaaaa" "aa"

Output: 4

Test Cases 3:

Input: "abababab" "aba"

Output: 3

Test Cases 4:

Input: "hello" "ll"

Output: 1

Test Cases 5:

Input: "abc" "d"

Output: 0

2) Preet's Anagram Mystery:

In the vibrant city of Jaipur, Rajasthan, renowned for its majestic palaces and rich cultural heritage, lives Preet—a young and enthusiastic programmer with a passion for solving intriguing problems. One evening, as the sun sets behind the stunning Hawa Mahal, Preet is presented with a captivating challenge.

The task is to determine whether two given strings are anagrams of each other. Anagrams are words or phrases formed by rearranging the letters of another, using all the original letters exactly once. This problem requires a keen eye for detail and a thorough understanding of string manipulation.

Preet, inspired by the beauty of her surroundings, sets up her laptop by the window, where she can see the twinkling lights of Jaipur. She begins to think about how to approach this problem efficiently, considering the constraints and ensuring the solution is both elegant and effective.

With a methodical approach, Preet envisions the strings as jumbled pieces of a jigsaw puzzle. Her goal is to rearrange the letters and determine if they fit together perfectly, forming an anagram. She decides to write a Python program that will check if the two strings contain the same letters with the same frequency, making them anagrams.

Join Preet on her journey to solve the anagram mystery. Witness how her logical thinking and programming skills come together to unravel the hidden connections within the strings. Let’s embark on this adventure of discovery and problem-solving with Preet, set against the enchanting backdrop of Jaipur.

Input Format:

- Two strings str1 and str2, space-separated. ($1 \leq |str1|, |str2| \leq 1000$)

Output Format:

- Print True if the strings are anagrams, otherwise print False.

Constraints:

- Both str1 and str2 contain only lowercase letters.
- Consider empty strings as not being anagrams of any other string.

Examples:

Example 1:
Input:
"listen" "silent"
Output:
True
Explanation:
The characters of "listen" can be rearranged to form "silent".

Example 2:
Input:
"hello" "world"
Output:
False
Explanation:
The characters of "hello" cannot be rearranged to form "world".

Test Cases:

Test Case 1:

Input: "triangle" "integral"
Output: True

Test Case 2:

Input: "abcd" "dcba"
Output: True

Test Case 3:

Input: "apple" "pale"
Output: False

Test Case 4:

Input: "aabbcc" "bbaacc"
Output: True

Test Case 5:

Input: "abc" "def"
Output: False

3) Victoria's Parisian Pangram Puzzle:

In the heart of Paris, where the Eiffel Tower stands tall and the Seine River flows gracefully, lives Victoria—a linguistics enthusiast with a love for languages and puzzles. One charming afternoon, as she sips her café au lait at a quaint Parisian café, Victoria encounters an intriguing challenge.

The task is to determine whether a given sentence is a pangram—a sentence that contains every letter of the English alphabet at least once. This problem requires a keen understanding of the alphabet and a methodical approach to string analysis.

With her laptop open on the café table, Victoria begins to think about how to approach this puzzle. The hustle and bustle of Paris provide the perfect backdrop for her intellectual pursuit. Her goal is to write a Python program that will check if the sentence is a pangram, ignoring spaces and punctuation.

As Victoria starts to code, she envisions the sentence as a canvas and the letters of the alphabet as colors that need to be present to complete the masterpiece. She meticulously analyzes each character in the sentence, ensuring that every letter from 'a' to 'z' is accounted for.

Join Victoria on her Parisian pangram puzzle adventure as she navigates the intricacies of string manipulation. Witness the elegance of her code as she transforms a simple task into an engaging exploration of language. Let’s embark on this journey with Victoria, appreciating the beauty of problem-solving in the enchanting city of Paris.

Examples:

Example 1:
Input:
"the quick brown fox jumps over a lazy dog"
Output:
True
Explanation:
The sentence contains all the letters from 'a' to 'z'.

Example 2:
Input:
"hello world"
Output:
False
Explanation:
The sentence does not contain all the letters from 'a' to 'z'.

Test Cases:

Test Case 1:

Input: "the five boxing wizards jump quickly"
Output: True

Test Case 2:

Input: "abc def ghi jkl mno pqr stu vwx yz"
Output: True

Test Case 3:

Input: "this is not a pangram"
Output: False

Test Case 4:

Input: "we promptly judged antique ivory buckles for the next prize"
Output: True

Test Case 5:

Input: "abcdefghijklmnopqrstuvwxyz"
Output: True

4) Ashutosh's Sacred Substring Search:

In the historic town of Sambhal, Uttar Pradesh, known for its rich cultural heritage, lives Ashutosh—a dedicated programmer with a passion for solving complex puzzles. One serene evening, as the sun sets over the town, Ashutosh finds himself facing an intriguing challenge.

The task is to find the length of the longest substring in a given string that contains at most K unique characters. This problem requires a keen understanding of string manipulation and the ability to identify patterns within text.

Inspired by the peaceful surroundings of Sambhal, Ashutosh sets up his laptop in his cozy room, with the distant sounds of evening prayers creating a tranquil ambiance. He begins to think about how to approach this problem efficiently, considering the constraints and ensuring the solution is both elegant and effective.

As he delves into the problem, Ashutosh recalls the teachings of God Kalki, the prophesied 10th avatar of Vishnu, who is said to bring an end to the current age of darkness and chaos. The thought of Kalki's wisdom and strength motivates him to tackle the challenge with clarity and determination.

Visualizing the string as a journey through the sacred streets of Sambhal, Ashutosh sees each unique character as a distinct landmark. His goal is to find the longest path that passes through at most K unique landmarks, much like navigating through the diverse and historic sites of his beloved town.

Join Ashutosh on his sacred substring search as he uncovers the longest substring with at most K unique characters. Witness how his logical thinking and programming skills, inspired by the divine wisdom of God Kalki, guide him through this intellectual adventure. Let’s embark on this journey of discovery and problem-solving with Ashutosh in the serene setting of Sambhal.

Input Format:
A string text ($1 \leq |\text{text}| \leq 1000$) and an integer K ($1 \leq K \leq 26$).

Output Format:
An integer representing the length of the longest substring.

Constraints:

- The string consists of only lowercase letters.

Examples:

Example 1:
Input:
"araaci" 2
Output:
4
Explanation:
The longest substring with at most 2 unique characters is "araa".

Example 2:
Input:
"araaci" 1
Output:
2
Explanation:
The longest substring with at most 1 unique character is "aa".

Test Cases:

Input: "abcabcbb" 2
Output: 2

Input: "aaa" 1
Output: 3

Input: "eceba" 2
Output: 3

Input: "aabbcc" 3
Output: 6

Input: "abcde" 4
Output: 4

5) Rahul's Palindromic Puzzle:

In the historic city of Agra, Uttar Pradesh, renowned for the timeless beauty of the Taj Mahal, lives Rahul—a passionate programmer with a love for solving intriguing problems. One calm evening, as the sun sets behind the majestic monument of love, Rahul finds himself faced with a fascinating challenge.

The task is to find the longest palindromic substring within a given string. A palindrome is a sequence that reads the same backward as forward, and this problem requires a meticulous approach to uncover the hidden symmetry within the text.

Inspired by the elegance of the Taj Mahal, Rahul sets up his laptop with the beautiful monument visible through his window. He begins to think about how to approach this problem efficiently, considering the constraints and ensuring the solution is both effective and elegant.

With a methodical mindset, Rahul visualizes the string as a tapestry of characters, each thread holding the potential for mirrored symmetry. His goal is to identify the longest sequence that maintains this harmonious balance, much like the architectural marvel he admires.

Join Rahul on his palindromic puzzle quest as he navigates through the complexities of string manipulation. Witness how his logical thinking and programming skills come together to reveal the longest palindromic substring hidden within the text. Let's embark on this journey of discovery and problem-solving with Rahul in the enchanting city of Agra.

Input Format:

- A string text ($1 \leq |text| \leq 1000$).

Output Format:

- A string representing the longest palindromic substring.

Constraints:

- The string consists of only lowercase letters.

Examples:

Example 1:
Input:
"babad"
Output:
"bab"
Explanation:
The longest palindromic substring is "bab" (or "aba" as both are valid).

Example 2:
Input:
"cbbd"
Output:
"bb"
Explanation:
The longest palindromic substring is "bb".

Test Cases:

Input: "forgeeksskeegfor"
Output: "geeksskeeg"

Input: "abc"
Output: "a"

Input: "aaaabbaa"
Output: "aabbaa"

Input: "a"
Output: "a"

Input: "racecar"
Output: "racecar"

6) Aditya's Palindromic Puzzle with Mahadev's Wisdom

In the serene coastal town of Konark, Odisha, famous for its magnificent Sun Temple, lives Aditya—a brilliant programmer with a passion for solving complex puzzles. One breezy evening, as the waves of the Bay of Bengal gently lap against the shore, Aditya finds himself faced with an intriguing challenge.

The task is to determine if the characters of a given string can be rearranged to form a palindrome. A palindrome is a sequence that reads the same backward as forward, and this problem requires a keen eye for detail and an understanding of character frequency.

Inspired by the divine presence of Mahadev, the great Shiva, Aditya sets up his laptop on the balcony of his home, with the majestic Sun Temple in view. He begins to think about how to approach this problem efficiently, ensuring the solution is both elegant and effective.

As Aditya delves into the task, he recalls the teachings of Mahadev—known for bringing balance and harmony to the universe. This thought inspires him to tackle the challenge with clarity and determination, seeking to find the perfect balance within the string.

Visualizing the string as a mosaic of characters, Aditya aims to determine whether the pieces can be rearranged symmetrically to form a perfect palindrome. His goal is to write a Python program that will check if the characters can be rearranged to achieve this harmonious balance.

Join Aditya on his palindromic puzzle quest as he navigates through the intricacies of string manipulation. Witness how his logical thinking and programming skills, inspired by the wisdom of Mahadev, guide him through this intellectual adventure. Let's embark on this journey of discovery and problem-solving with Aditya in the enchanting town of Konark.

Input Format:

- A single string text containing only lowercase alphabetic characters.
  $(1 \leq |\text{text}| \leq 1000)$

Output Format:

- Output True if the string's characters can be rearranged to form a palindrome, otherwise output False.

Constraints:

- The input string will only contain lowercase alphabetic characters.
- The input string does not include spaces or special characters.

Examples:

Example 1:
Input:
"civic"
Output:
True
Explanation:
The string "civic" is already a palindrome.

Example 2:
Input:
"aabbcc"
Output:
True
Explanation:
The string can be rearranged to form "abccba", which is a palindrome.

Test Cases:

Input: "abcde"
Output: False

Input: "aabbccd"
Output: True

Input: "racecar"
Output: True

```
Input: "levelup"
Output: False

Input: "nono"
Output: True
```

List

Ques-1

Scenario: You are a software developer at a gaming company. Your manager has given you a task related to a leaderboard system. Players earn points after completing various levels, and the scores are stored in a list. Your task is to identify the second-highest score from the list to award the second-place player. The list of scores may contain duplicates, but only unique scores should be considered for determining the second-highest score.

Constraints:

1. If the list has less than two unique scores, there should be no second-highest score, and the result should be "No second-highest score."
2. The input list will always contain integers.

Sample Input and Output with Explanation:

1. **Input:**
   [2, 3, 1, 4, 5]
   Output:
   4

Explanation:
The unique scores are [1, 2, 3, 4, 5]. The highest score is 5, and the second-highest score is 4.

2. **Input:**
   [5, 5, 5, 5]
   Output:
   No second-highest score

Explanation:
The list has only one unique score [5]. Since there is no second-highest score, the output is "No second-highest score".

3. **Input:**
   [10, 40, 20, 20, 30, 30,40]
   Output:
   30

Explanation:
The unique scores are [10, 20, 30, 40]. The highest score is 40, and the second-highest score is 30.

Test Cases:

1. **Input:** [7, 10, 5, 8, 9]
   Output: 9
2. **Input:** [100, 200, 100, 50, 200]
   Output: 100
3. **Input:** [1]
   Output: No second-highest score
4. **Input:** [4, 1, 2, 4, 3]
   Output: 3
5. **Input:** [0, -1, -2, -1, 0]
   Output: -1

Ques-2

Scenario: You are working as a software engineer at a company that tracks inventory. Each product is assigned a unique number in a consecutive series. However, due to a system glitch, one number is missing from the series. Your task is to identify the missing number so the inventory can be corrected. The series is guaranteed to have exactly one missing number, and all other numbers appear exactly once.

Constraints:

1. The input list will contain integers in ascending order.
2. The list will always have at least two numbers with one number missing from the series.

Sample Input and Output with Explanation:

1. **Input:**
   [1, 2, 4, 5]
   Output:
   3
   Explanation:
   The series [1, 2, 3, 4, 5] is expected, but 3 is missing.
2. **Input:**
   [10, 11, 12, 14]
   Output:
   13
   Explanation:
   The series [10, 11, 12, 13, 14] is expected, but 13 is missing.
3. **Input:**
   [100, 101, 102, 103, 105]
   Output:
   104
   Explanation:
   The series [100, 101, 102, 103, 104, 105] is expected, but 104 is missing.

Test Cases:

1. **Input:** [1, 2, 3, 5]
   Output: 4
2. **Input:** [20, 21, 22, 24]
   Output: 23

3. **Input:** [50, 51, 52, 54]
   Output: 53
4. **Input:** [7, 8, 10]
   Output: 9
5. **Input:** [15, 16, 17, 19]
   Output: 18

Ques-3

Scenario: Rotate a list by k elements

A conveyor belt system moves items in a circular fashion. To align items for a specific process, you need to rotate the list of item positions to the right by k steps. Your task is to determine the new order of items after the rotation.

Sample Input and Output with Explanation:

1. **Input:**
   list = [1, 2, 3, 4, 5], k = 2
   Output:
   [4, 5, 1, 2, 3]
   Explanation:
   Rotating the list [1, 2, 3, 4, 5] to the right by 2 positions gives [4, 5, 1, 2, 3].
2. **Input:**
   list = [10, 20, 30, 40, 50], k = 3
   Output:
   [30, 40, 50, 10, 20]
   Explanation:
   Rotating [10, 20, 30, 40, 50] by 3 positions gives [30, 40, 50, 10, 20].
3. **Input:**
   list = [7, 8, 9], k = 1
   Output:
   [9, 7, 8]
   Explanation:
   Rotating [7, 8, 9] by 1 position gives [9, 7, 8].

Test Cases:

1. **Input:** list = [0, 1, 2, 3], k = 6
   Output: [2, 3, 0, 1]
2. **Input:** list = [1, 2, 3, 4, 5], k = 1
   Output: [5, 1, 2, 3, 4]
3. **Input:** list = [1, 2, 3, 4, 5], k = 5
   Output: [1, 2, 3, 4, 5]

4. **Input:** list = [1, 2, 3, 4, 5], k = 0
   Output: [1, 2, 3, 4, 5]
5. **Input:** list = [10, 20, 30], k = 2
   Output: [20, 30, 10]

Ques-4

Given an list of integers **nums** and an integer **target**, return *indices of the two numbers such that they add up to target*.

You may assume that each input would have ***exactly* one solution**, and you may not use the *same* element twice.

You can return the answer in any order.

Constraints:

- $2 <= \text{nums.length} <= 10^4$
- $-10^9 <= \text{nums[i]} <= 10^9$
- $-10^9 <= \text{target} <= 10^9$

Sample Input and Output with Explanation:

Example 1:

Input: nums = [2,7,11,15], target = 9

Output: [0,1]

Explanation: Because nums[0] + nums[1] == 9, we return [0, 1].

Example 2:

Input: nums = [3,2,4], target = 6

Output: [1,2]

Explanation: Because 1 and 2 are 2 and 4, which add up to the target 6.

Example 3:

Input: nums = [3,3], target = 6

Output: [0,1]

Explanation: Because 0 and 1 are 3 and 3, which add up to the target 6.

Test Cases:

1. **Input:** nums = [1, 2, 3, 4], target = 5
   Output: [0, 3]
2. **Input:** nums = [2, 5, 5, 11], target = 10
   Output: [1, 2]
3. **Input:** nums = [1, 7, 11, 15], target = 8
   Output: [0, 1]
4. **Input:** nums = [6, 3, 4, 8], target = 10
   Output: [1, 2]
5. **Input:** nums = [3, 5, 9, 2], target = 8
   Output: [0, 3]

Ques-5

Counting Words With a Given Prefix

You are given an list of strings words and a string pref.

Return *the number of strings in* words *that contain* pref *as a **prefix***.

A **prefix** of a string s is any leading contiguous substring of s.

Example 1:

Input: words = ["pay","**at**tention","practice","**at**tend"], pref = "at"
Output: 2
Explanation: The 2 strings that contain "at" as a prefix are: "**at**tention" and "**at**tend".

Example 2:

Input: words = ["leetcode","win","loops","success"], pref = "code"
Output: 0
Explanation: There are no strings that contain "code" as a prefix.

Test Cases

1. **Input:** words = ["cat", "car", "carbon", "dog"], pref = "ca"
   Output: 3
2. **Input:** words = ["hero", "hermit", "heritage", "hello"], pref = "her"
   Output: 3
3. **Input:** words = ["light", "like", "life", "line"], pref = "li"
   Output: 4
4. **Input:** words = ["zoom", "zebra", "zone"], pref = "zo"
   Output: 2
5. **Input:** words = ["book", "bookmark", "board"], pref = "boo"
   Output: 2

Ques-6

You are given a matrix A, you have to return an list containing sum of each row elements followed by sum of each column elements of the matrix.

NOTE: If the matrix given is of size (N x M), then the array you return would be of size **(N + M)**, where first N elements contain the sum of each N rows, and the next M elements contain the sum of each M columns.

Example

Input [[1, 2],[4, 5],[8, 9]]
Output [3, 9, 17, 13, 16]
Explanation SUM OF ROWS ARE 3,9,17

SUM OF COLUMNS ARE 13,16

Test Case

1. [[1, 2],[4, 5],[8, 9]]
[3, 9, 17, 13, 16]

2. [[1,4,2],[8,0,3]]
[7, 11, 9, 4, 5]

3. [[1,2], [6,4], [9,5], [0,7]]
[3, 10, 14, 7, 16, 18]

4. [[2,4,8,1],[0,5,3,7]]
[15, 15, 2, 9, 11, 8]

Tuple

Question 1:

Scenario: You are a software developer at a gaming company. Your manager has given you a task related to a leaderboard system. Players earn points after completing various levels, and the scores are stored in a list. Your task is to identify the second-highest score from the list to award the second-place player. The list of scores may contain duplicates, but only unique scores should be considered for determining the second-highest score.

Sample Input and Output with Explanation:

Input : scores = (85, 90, 78, 92, 88, 76, 95)

Output: Average Score: 86.57

Explanation:

In this case, the scores tuple contains the following values:
85, 90, 78, 92, 88, 76, 95. To calculate the average:

- Add all the scores: 85 + 90 + 78 + 92 + 88 + 76 + 95 = 604
- Divide the total by the number of students: 604 / 7 = 86.57 .

Input : scores = (65, 70, 75, 80, 85)

Output : Average Score: 75.0

Explanation:

For this example, the scores are: 65, 70, 75, 80, 85.

- Add all the scores: 65 + 70 + 75 + 80 + 85 = 375
- Divide by the number of students: 375 / 5 = 75.0

Test Cases:

1. **Input**: (45, 50, 55, 60, 65)
   Output: Average Score: 55.0

2. **Input**: (100, 95, 90, 85, 80, 75, 70, 65)
   Output: Average Score: 82.5
3. **Input**: (50, 60, 70)
   Output: Average Score: 60.0
4. **Input**: (30, 40, 50, 60, 70, 80, 90)
   Output: Average Score: 60.0
5. **Input**: (10, 20, 30, 40)
   Output: Average Score: 25.0

Question 2:

Scenario: You have been given a tuple containing a list of products sold in a store. However, some of these products have been listed multiple times. As part of the inventory management system, you need to remove any duplicates from the tuple to get a list of unique products. Your task is to write a function that returns a new tuple with only the unique product names.

Example Input 1:

products = ("apple", "banana", "orange", "banana", "apple", "grape", "orange")

Expected Output 1:('apple', 'banana', 'orange', 'grape')

Explanation: In this case, the tuple contains the following products: ("apple", "banana", "orange", "banana", "apple", "grape", "orange"). To remove duplicates:

- The first occurrence of "apple" remains, and the second occurrence is removed.
- The first occurrence of "banana" remains, and the second is removed.
- The first occurrence of "orange" remains, and the second is removed.
- "grape" remains as it appears only once. Thus, the result will be ('apple', 'banana', 'orange', 'grape').

Example Input 2:

products = ("laptop", "phone", "tablet", "laptop", "camera", "phone")

Expected Output 2:

('laptop', 'phone', 'tablet', 'camera')

Explanation:

In this case, the tuple contains the following products: ("laptop", "phone", "tablet", "laptop", "camera", "phone"). To remove duplicates:

- The first occurrence of "laptop" remains, and the second is removed.

- The first occurrence of "phone" remains, and the second is removed.
- "tablet" and "camera" are unique, so they are included as well. Thus, the result will be ('laptop', 'phone', 'tablet', 'camera').

Test Cases:

1. **Input**: ("cat", "dog", "cat", "bird", "dog", "fish")
   Output: ('cat', 'dog', 'bird', 'fish')
2. **Input**: ("red", "blue", "green", "blue", "yellow", "green")
   Output: ('red', 'blue', 'green', 'yellow')
3. **Input**: ("book", "pen", "book", "pencil", "pen")
   Output: ('book', 'pen', 'pencil')
4. **Input**: ("apple", "banana", "apple", "orange", "kiwi", "orange")
   Output: ('apple', 'banana', 'orange', 'kiwi')
5. **Input**: ("milk", "bread", "butter", "cheese", "milk", "butter")
   Output: ('milk', 'bread', 'butter', 'cheese')

Question 3

Scenario: You are working on an employee management system where you are storing employee names in a tuple. One day, a manager asks for the position of a specific employee within the list. Your task is to write a function that finds the index of a given employee name in the tuple. The function should return the index if the employee is present in the tuple, and it should return -1 if the employee is not found.

Example Input 1:

employees = ("John", "Alice", "Bob", "Sarah", "Tom")

Expected Output 1:

Index of 'Bob': 2

Explanation:

In this case, the tuple contains the employee names:
("John", "Alice", "Bob", "Sarah", "Tom").

- "John" is at index 0.
- "Alice" is at index 1.
- "Bob" is at index 2, so the function returns 2.

Example Input 2:

employees = ("Michael", "Anna", "James", "Emma")

Expected Output 2:

Index of 'James': 2

Explanation:

In this case, the tuple contains the employee names:
("Michael", "Anna", "James", "Emma").

- "Michael" is at index 0.

- "Anna" is at index 1.
- "James" is at index 2, so the function returns 2.

Test Cases:

1. **Input**: ("George", "Sophia", "William", "Emma")
   Output: Index of 'Sophia': 1
2. **Input**: ("Daniel", "Sophia", "Paul", "Lucas")
   Output: Index of 'Lucas': 3
3. **Input**: ("Eve", "Grace", "Henry", "Zoe")
   Output: Index of 'Zoe': 3
4. **Input**: ("Ava", "Ella", "Liam", "Noah")
   Output: Index of 'Liam': 2
5. **Input**: ("Jack", "Emma", "Mason", "Olivia")
   Output: Index of 'Zara': -1

Question 4

Scenario: You are developing an e-commerce system where a tuple contains the product IDs of items in a customer's shopping cart. However, the system needs to update the cart by adding or removing products dynamically. Since tuples are immutable, you need to convert the tuple to a list, which will allow modifications (such as adding or removing product IDs). Your task is to write a function that converts a given tuple of product IDs into a list.

Example Input 1:

cart_items = (101, 102, 103, 104)

Expected Output 1:

[101, 102, 103, 104]

Explanation:

In this case, the cart contains the following product IDs:
(101, 102, 103, 104).
After converting the tuple to a list, the result will be [101, 102, 103, 104].

Example Input 2:

cart_items = (201, 202, 203)

Expected Output 2:

[201, 202, 203]

Explanation:

Here, the cart contains the product IDs:
(201, 202, 203).
After converting to a list, the result is [201, 202, 203].

Test Cases:

1. **Input**: (301, 302, 303, 304)
   Output: [301, 302, 303, 304]
2. **Input**: (501, 502)
   Output: [501, 502]
3. **Input**: (1001, 1002, 1003)
   Output: [1001, 1002, 1003]
4. **Input**: (11, 22, 33)
   Output: [11, 22, 33]
5. **Input**: (1, 2, 3, 4, 5)
   Output: [1, 2, 3, 4, 5]

Question 5

Scenario: You are working on a customer database system where customer IDs are stored in two separate tuples. These tuples contain unique customer IDs from two different departments. However, some customers might have been added in both departments' lists, resulting in duplicate IDs. Your task is to merge these two tuples into one, remove any duplicate customer IDs, and return the result as a new tuple.

Example Input 1:

department_a = (101, 102, 103, 104)

department_b = (103, 105, 106)

Expected Output 1:

(101, 102, 103, 104, 105, 106)

Explanation:

In this case, the customer IDs from both departments are:
department_a = (101, 102, 103, 104)
department_b = (103, 105, 106).

- The ID "103" appears in both tuples, so it should appear only once in the merged result.
- After merging and removing duplicates, the final result is: (101, 102, 103, 104, 105, 106).

Example Input 2:

department_a = (201, 202, 203)

department_b = (203, 204, 205)

Expected Output 2:

(201, 202, 203, 204, 205)

Explanation:

In this case, the customer IDs from both departments are:
department_a = (201, 202, 203)
department_b = (203, 204, 205).

- The ID "203" appears in both tuples, so it should appear only once.
- After merging and removing duplicates, the result is: (201, 202, 203, 204, 205).

Test Cases:

1. **Input**: (111, 112, 113, 114), (113, 115, 116)
   Output: (111, 112, 113, 114, 115, 116)
2. **Input**: (301, 302, 303, 304), (305, 302, 306)
   Output: (301, 302, 303, 304, 305, 306)
3. **Input**: (501, 502, 503), (503, 504, 505)
   Output: (501, 502, 503, 504, 505)
4. **Input**: (1001, 1002), (1003, 1004, 1001)
   Output: (1001, 1002, 1003, 1004)
5. **Input**: (121, 122, 123), (123, 124, 125)
   Output: (121, 122, 123, 124, 125)

Question 6

Scenario: You are working on a sports tournament management system where players from two different teams have been registered. These players' IDs are stored in two separate tuples. The tournament committee wants to identify which players are part of both teams, as these players might have transferred between teams. Your task is to find and return the common player IDs (i.e., the players who exist in both tuples).

Example Input 1:

team_a = (101, 102, 103, 104)

team_b = (103, 105, 106)

Expected Output 1:

(103)

Explanation:

In this case, the player IDs from both teams are:
team_a = (101, 102, 103, 104)
team_b = (103, 105, 106).

- The player ID "103" is present in both teams, so it is the common element between the two tuples.

Example Input 2:

team_a = (201, 202, 203)

team_b = (203, 204, 205)

Expected Output 2:

(203)

Explanation:

In this case, the player IDs from both teams are:
team_a = (201, 202, 203)
team_b = (203, 204, 205).

- The player ID "203" is present in both teams, so it is the common element between the two tuples.

Test Cases:

1. **Input**: (111, 112, 113, 114), (113, 115, 116)
   Output: (113)
2. **Input**: (301, 302, 303, 304), (305, 302, 306)
   Output: (302)
3. **Input**: (501, 502, 503), (503, 504, 505)
   Output: (503)
4. **Input**: (1001, 1002), (1003, 1004, 1001)
   Output: (1001)
5. **Input**: (121, 122, 123), (123, 124, 125)
   Output: (123)

Dictionary

Scenario 1:

Retrieve Contact Details
In this program, you are provided with a dictionary containing the name and phone number of a contact. Your task is to retrieve and display the phone number for a given name. This question tests your ability to access values from a dictionary in Python.

Input Format:

- A single name of a contact (string).

Constraints:

- The dictionary contains only one key-value pair: {"Alice": "9876543210"}.
- The input name will always exist in the dictionary.

Output Format:

- A single line with the format: "Phone number of [Name]: [Phone Number]".

Sample Inputs and Outputs:

Sample Input	Sample Output
Alice	Phone number of Alice: 9876543210
Ram	Phone number of Ram: 9876513210
Sita	Phone number of Alice: 9819543210
Tarun	Phone number of Alice: 9876273210
Naman	Phone number of Alice: 8276543210

Scenario 2:

Access Value from Dictionary

A dictionary in Python stores data in key-value pairs, where keys are unique, and values are associated with these keys. You can retrieve a value by specifying its corresponding key using the syntax dictionary[key].

This program allows you to input a dictionary and a key to retrieve the associated value.

Input Format:

- A dictionary in JSON-like format.
- A key whose value needs to be retrieved.

Constraints:

- The key must exist in the dictionary.

Output Format:

- The value corresponding to the input key.

Sample Inputs and Outputs:

Sample Input	Sample Output
{"name": "Alice", "age": 25} name	Alice
{"a": 10, "b": 20, "c": 30} b	20
{"fruit": "apple", "color": "red"} color	red
{"x": 5, "y": 15} y	15
{"language": "Python"} language	Python

Scenario 3:

Calculate Average Marks
You are required to calculate the average marks for a student based on their scores in three subjects: Math, Science, and English. Additionally, you will add a History subject with predefined marks to the student's record. This problem helps you learn how to update a dictionary and perform basic arithmetic operations.

Input Format:

- Three space-separated integers representing marks for Math, Science, and English.

Constraints:

- $0 \leq \text{Marks} \leq 100$.

Output Format:

- Two lines:
  1. The updated dictionary (with History marks).
  2. The average marks rounded to 2 decimal places.

Sample Inputs and Outputs:

Sample Input	**Sample Output**
80 90 85	Updated Marks: {'Math': 80, 'Science': 90, 'English': 85, 'History': 88} Average Marks: 85.75
70 85 80	Updated Marks: {'Math': 70, 'Science': 85, 'English': 80, 'History': 88} Average Marks: 80.75
95 88 92	Updated Marks: {'Math': 95, 'Science': 88, 'English': 92, 'History': 88} Average Marks: 90.75
60 75 65	Updated Marks: {'Math': 60, 'Science': 75, 'English': 65, 'History': 88} Average Marks: 72.00
50 50 50	Updated Marks: {'Math': 50, 'Science': 50, 'English': 50, 'History': 88} Average Marks: 59.50

Scenario 4:

Employee Dictionary Manipulation

Theory:
This task involves creating an employee dictionary from user input, where each employee has an ID and a name. Additionally, you will create a second dictionary that stores the length of each employee's name. This problem strengthens your knowledge of dictionary manipulations and basic string operations.

Input Format:

- An integer N (number of employees).
- N lines containing space-separated employee ID (integer) and name (string).

Constraints:

- 1 ≤ N ≤ 100.
- Employee name length ≤ 50.

Output Format:

- The employee dictionary.
- The dictionary of employee name lengths.

Sample Inputs and Outputs:

Sample Input	Sample Output

3
101 Alice
102 Bob
103 Charlie
	Employees: {101: 'Alice', 102: 'Bob', 103: 'Charlie'}
	Name Lengths: {101: 5, 102: 3, 103: 7}

2
201 David
202 Edward
	Employees: {201: 'David', 202: 'Edward'}
	Name Lengths: {201: 5, 202: 6}

4
301 Grace
302 Helen
303 Irene
	Employees: {301: 'Grace', 302: 'Helen', 303: 'Irene', 304: 'Julia'}
	Name Lengths: {301: 5, 302: 5, 303: 5, 304: 5}

304 Julia	

1
401 Frank
	Employees: {401: 'Frank'}
	Name Lengths: {401: 5}

5
501 Anne
502 Bea
503 Chloe
504 Diana
505 Elle
	Employees: {501: 'Anne', 502: 'Bea', 503: 'Chloe', 504: 'Diana', 505: 'Elle'}
	Name Lengths: {501: 4, 502: 3, 503: 5, 504: 5, 505: 4}

Scenario 5:

Filter Dictionary Using Comprehension

Dictionary comprehensions allow creating dictionaries from existing ones based on a specific condition. You can use {key: value for key, value in original_dict.items() if condition} for filtering.

This task involves filtering a dictionary by retaining only those key-value pairs where the value is greater than a given threshold.

Input Format:

- A dictionary with integer values.
- A threshold value ttt.

Constraints:

- The dictionary will contain only integer values.
- Only keys with values greater than ttt should remain.

Output Format:

- A dictionary containing the filtered key-value pairs.

Sample Inputs and Outputs:

Sample Input	Sample Output
{"a": 10, "b": 20, "c": 5} 15	{"b": 20}
{"x": 100, "y": 200, "z": 50} 150	{"y": 200}
{"apple": 3, "banana": 7} 5	{"banana": 7}
{"p": 1, "q": 5, "r": 10} 0	{"p": 1, "q": 5, "r": 10}
{"math": 75, "science": 80} 90	{}