# Roman Numeral Formatter

\*\*Description:\*\* Convert a given integer to a Roman numeral for use in certificates, badges, or version branding.

\*\*Problem Statement:\*\*  
In various formal and creative domains—like certificate generation, historic data modeling, or version tagging for premium content—representing numbers in Roman numeral format is often preferred for stylistic or traditional value.  
  
You are tasked with implementing a system component that takes in a modern decimal number and formats it into its Roman numeral equivalent.  
  
For example, you may have to label a certificate as "Edition IV", "Version IX", or "Chapter XVI", depending on the integer input.  
  
The Roman numeral system uses combinations of letters from the Latin alphabet:  
- I (1), V (5), X (10), L (50), C (100), D (500), M (1000)  
- Subtractive notation is used, e.g., IV (4), IX (9), XL (40), etc.  
  
Your task is to convert an integer from 1 to 3999 into its correct Roman numeral representation, ensuring that the output follows standard Roman numeral rules.

\*\*Input Format:\*\*  
A single line containing an integer num representing the value to convert.

\*\*Constraints:\*\*  
1 <= num <= 3999

\*\*Output Format:\*\*  
Print a single line containing the Roman numeral equivalent of the given number.

\*\*Tags:\*\* conversion, strings, roman numeral, greedy, branding, intermediate

\*\*Sample Input:\*\*  
1994

\*\*Sample Output:\*\*  
MCMXCIV

\*\*Test Cases:\*\*

* Test Case 1 Input:  
  1994
* Expected Output:  
  MCMXCIV
* Test Case 2 Input:  
  58
* Expected Output:  
  LVIII
* Test Case 3 Input:  
  4
* Expected Output:  
  IV
* Test Case 4 Input:  
  9
* Expected Output:  
  IX
* Test Case 5 Input:  
  3999
* Expected Output:  
  MMMCMXCIX
* Test Case 6 Input:  
  2024
* Expected Output:  
  MMXXIV
* Test Case 7 Input:  
  1
* Expected Output:  
  I
* Test Case 8 Input:  
  1000
* Expected Output:  
  M
* Test Case 9 Input:  
  944
* Expected Output:  
  CMXLIV
* Test Case 10 Input:  
  1666
* Expected Output:  
  MDCLXVI

# Isomorphic Identifier Mapping

\*\*Description:\*\* Check if two strings follow the same character mapping pattern — useful in compilers or encryption.

\*\*Problem Statement:\*\*  
In scenarios like compiler design, interpreters, encryption engines, or symbolic translation tools, it is critical to check whether two sets of identifiers or tokens follow the same mapping structure.  
  
Two strings s and t are said to be isomorphic if the characters in s can be replaced to get t, with a one-to-one and consistent mapping between characters.  
  
This means:  
- Each character in s maps to a single character in t.  
- No two characters in s map to the same character in t.  
- The order and repetition pattern must remain the same.  
  
Your task is to determine whether two strings s and t are isomorphic.

\*\*Input Format:\*\*  
The first line contains string s.  
The second line contains string t.

\*\*Constraints:\*\*  
1 <= len(s), len(t) <= 10^4  
s and t contain only lowercase English letters.

\*\*Output Format:\*\*  
Print True if the strings are isomorphic, otherwise print False.

\*\*Tags:\*\* strings, hashmap, isomorphism, compiler, encryption, pattern matching, intermediate

\*\*Sample Input:\*\*  
egg  
dd

\*\*Sample Output:\*\*  
True

\*\*Test Cases:\*\*

* Test Case 1 Input:  
  egg  
  add
* Expected Output:  
  True
* Test Case 2 Input:  
  foo  
  bar
* Expected Output:  
  False
* Test Case 3 Input:  
  paper  
  title
* Expected Output:  
  True
* Test Case 4 Input:  
  badc  
  baba
* Expected Output:  
  False
* Test Case 5 Input:  
  abc  
  def
* Expected Output:  
  True
* Test Case 6 Input:  
  aab  
  xyz
* Expected Output:  
  False
* Test Case 7 Input:  
  ab  
  aa
* Expected Output:  
  False
* Test Case 8 Input:  
  turtle  
  tletur
* Expected Output:  
  False
* Test Case 9 Input:  
  abcd  
  efgh
* Expected Output:  
  True
* Test Case 10 Input:  
  abab  
  cdcd
* Expected Output:  
  True

# IPv4 Address Validator

\*\*Description:\*\* Validate if a given string is a valid IPv4 address using regex, ensuring each octet is within the 0–255 range.

\*\*Problem Statement:\*\*  
In networking tools, cybersecurity platforms, and system configurations, it is crucial to validate whether user input is a properly formatted and valid IPv4 address. An IPv4 address consists of four decimal numbers (octets) separated by dots (.), where each number ranges from 0 to 255.  
  
For example, 192.168.1.1 is a valid IPv4 address, while 256.300.888.1, 192.168.01.1, and 192.168.1 are not.  
  
Your task is to write a program that uses regular expressions (regex) to verify if a given input string qualifies as a valid IPv4 address.  
  
Rules:  
- Must contain exactly four integers separated by three dots.  
- Each integer (octet) must be in the range 0 to 255, inclusive.  
- Leading zeros in octets are not allowed unless the octet is exactly 0 (i.e., 01, 001 are invalid).  
- Input should contain only digits and dots — no extra characters.

\*\*Input Format:\*\*  
A single line containing the string ip — the IP address to validate.

\*\*Constraints:\*\*  
The input string contains only printable ASCII characters.  
1 <= len(ip) <= 20

\*\*Output Format:\*\*  
Print True if the input is a valid IPv4 address, otherwise print False.

\*\*Tags:\*\* regex, validation, string matching, ipv4, networking, intermediate

\*\*Sample Input:\*\*  
192.168.1.1

\*\*Sample Output:\*\*  
True

\*\*Test Cases:\*\*

* Test Case 1 Input:  
  192.168.1.1
* Expected Output:  
  True
* Test Case 2 Input:  
  255.255.255.255
* Expected Output:  
  True
* Test Case 3 Input:  
  0.0.0.0
* Expected Output:  
  True
* Test Case 4 Input:  
  256.100.100.100
* Expected Output:  
  False
* Test Case 5 Input:  
  192.168.1
* Expected Output:  
  False
* Test Case 6 Input:  
  192.168.1.01
* Expected Output:  
  False
* Test Case 7 Input:  
  192.168.1.1.1
* Expected Output:  
  False
* Test Case 8 Input:  
  abc.def.gha.bcd
* Expected Output:  
  False
* Test Case 9 Input:  
  172.16.254.1
* Expected Output:  
  True
* Test Case 10 Input:  
  1.2.3.256
* Expected Output:  
  False