

Scenario 1: User Input Validation A website accepts user input in the form of an email address. Your task is to write a function that checks if the input string is a valid email address. The email should contain only one "@" symbol, contain at least one character before the "@" symbol, and end with a valid domain such as ".com", ".org", or ".edu".

Hints:

Use `str.count('@')` to ensure there's exactly one "@".

Use `str.split('@')` to check for the presence of text before and after the "@".

Use `str.endswith()` to verify the domain suffix.

Scenario 2: Sorting Words in a Sentence A user inputs a sentence, and you need to sort the words in the sentence alphabetically. The words are separated by spaces, and punctuation should not interfere with the sorting.

Hints:

Use `split()` to break the sentence into words.

Use `sorted()` to sort the words alphabetically.

Use `str.strip()` or regular expressions to remove punctuation from words before sorting.

Scenario 3: Reversing a Substring You are given a string and two indices (start and end). Your task is to reverse the substring between these indices and return the new string.

Hints:

Use slicing: `string[start:end]` to extract the substring.

Reverse a substring using slicing: `substring[::-1]`.

Scenario 4: Anagram Checker You are given two strings, and your task is to check if one string is an anagram of the other. An anagram is a word or phrase formed by rearranging the letters of a different word.

Hints:

Sort both strings and compare if they are the same.

Use `sorted()` and check if the results are identical.

Alternatively, use a frequency counter (`collections.Counter`).

Scenario 5: Palindrome Checker Write a function to check whether a given string is a palindrome. A palindrome is a word that reads the same forward and backward.

Hints:

Use slicing: `string[::-1]` to reverse the string.

Compare the original string with the reversed string to check for equality.

Based on Lists

Scenario 1: Merging Two Sorted Lists You are given two sorted lists of integers. Your task is to merge them into a single sorted list.

Hints:

Use two pointers to iterate through both lists and insert elements into a result list in sorted order.

Alternatively, you can use the `sorted()` function on the merged list.

Scenario 2: Finding Duplicates You are given a list of integers and need to find

all the elements that appear more than once in the list.

Hints:

Use a dictionary (or `collections.Counter`) to count occurrences of each element. Extract elements that have a count greater than 1.

Scenario 3: Removing Odd Numbers You are given a list of integers, and you need to remove all odd numbers from the list.

Hints:

Use a list comprehension: `[num for num in lst if num % 2 == 0]`.

Scenario 4: List Rotation You are given a list and an integer `n`. Your task is to rotate the list to the right by `n` positions. For example, if the list is `[1, 2, 3, 4, 5]` and `n=2`, the result should be `[4, 5, 1, 2, 3]`.

Hints:

Use list slicing: `lst[-n:] + lst[:-n]`.

Handle cases where `n` is larger than the length of the list using modulo: `lst[-(n % len(lst)):] + lst[:- (n % len(lst))]`.

Scenario 5: Flattening a Nested List You are given a nested list (a list of lists) containing integers, and you need to flatten it into a single list of integers.

Hints:

Use list comprehension to iterate through the inner lists and concatenate the results: `[item for sublist in lst for item in sublist]`.

You could also use a recursive function to flatten the list.

Scenario 6: Find the Maximum Product of Two Numbers You are given a list of integers and need to find the pair of numbers that has the maximum product.

Scenario 7: Removing All Occurrences of a Specific Element You are given a list and a specific element. Write a function to remove all occurrences of that element from the list.